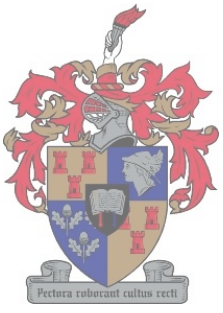
Ondersoek na die gebruik van CAN in 'n laespoed kommunikasie stelsel vir 'n volgende generasie SUNSAT mikro satelliet.



deur
C.H. Steyn

Tesis ingelewer ter gedeeltelike voldoening aan die vereistes vir die graad Magister van Wetenskap in Ingenieurswese aan die Universiteit van Stellenbosch

Studieleier: Prof. P.J. Bakkes
Desember 1999

Verklaring:

Ek, die ondergetekende verklaar hiermee dat die werk in hierdie tesis vervat, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik by enige universiteit ter verkryging van 'n graad voorgelê het nie.

Datum: 01-02-2000

Opsomming:

Die huidige SUNSAT 1 se kommunikasie struktuur bestaan uit parallelle interverbindings met 'n groot aantal unieke koppelvlakke. Alhoewel die stelsel goed funksioneer het dit sekere ongewenste eienskappe soos byvoorbeeld 'n hoë verwerkerlas. 'n Verdere probleem is dat 'n unieke drywer ontwikkel moet word vir elke koppelvlak.

Vir 'n volgende generasie SUNSAT mikro satelliet is die behoefte daar gestel om die kommunikasie struktuur te verbeter en te vervang met 'n netwerkstelsel.

In hierdie tesis word daar gekyk na hoe die CAN ("Controller Area Network") protokol gebruik kan word in 'n algemene kommunikasiestelsel. Verskillende opstellings word oorweeg en die eienskappe van die stelsels word bespreek.

Summary:

The communication structure of the current SUNSAT 1 micro satellite employs a parallel architecture, which consists of a number of different unique interfaces. The current system however has some undesirable properties, some of which are the high processor load and the amount of different interfaces that exist.

For a next generation SUNSAT micro satellite the need arose for a new communication structure, which replaces the parallel structure with a network system.

In this thesis the CAN (Controller Area Network) protocol is evaluated for implementation in a possible next generation communication system. Different systems are evaluated and the properties of each system are discussed.

Dankbetuiging:

Ek wil graag dank uitspreek teenoor die volgende persone waarsonder die voltooiing van hierdie tesis nie moontlik sou wees nie.

- My studieleier, Prof P.J. Bakkes vir sy waardevolle leiding.
- My vrou, Carmé, wat altyd bereid is om te luister.
- My ouers, vir die geleentheid wat hulle aan my gebied het om verder te studeer.
- My Almatige God wat alles moontlik maak.

Inhoudsopgawe:

Lys Van Figure:

Lys Van Tabelle:

Lys van Akronieme:

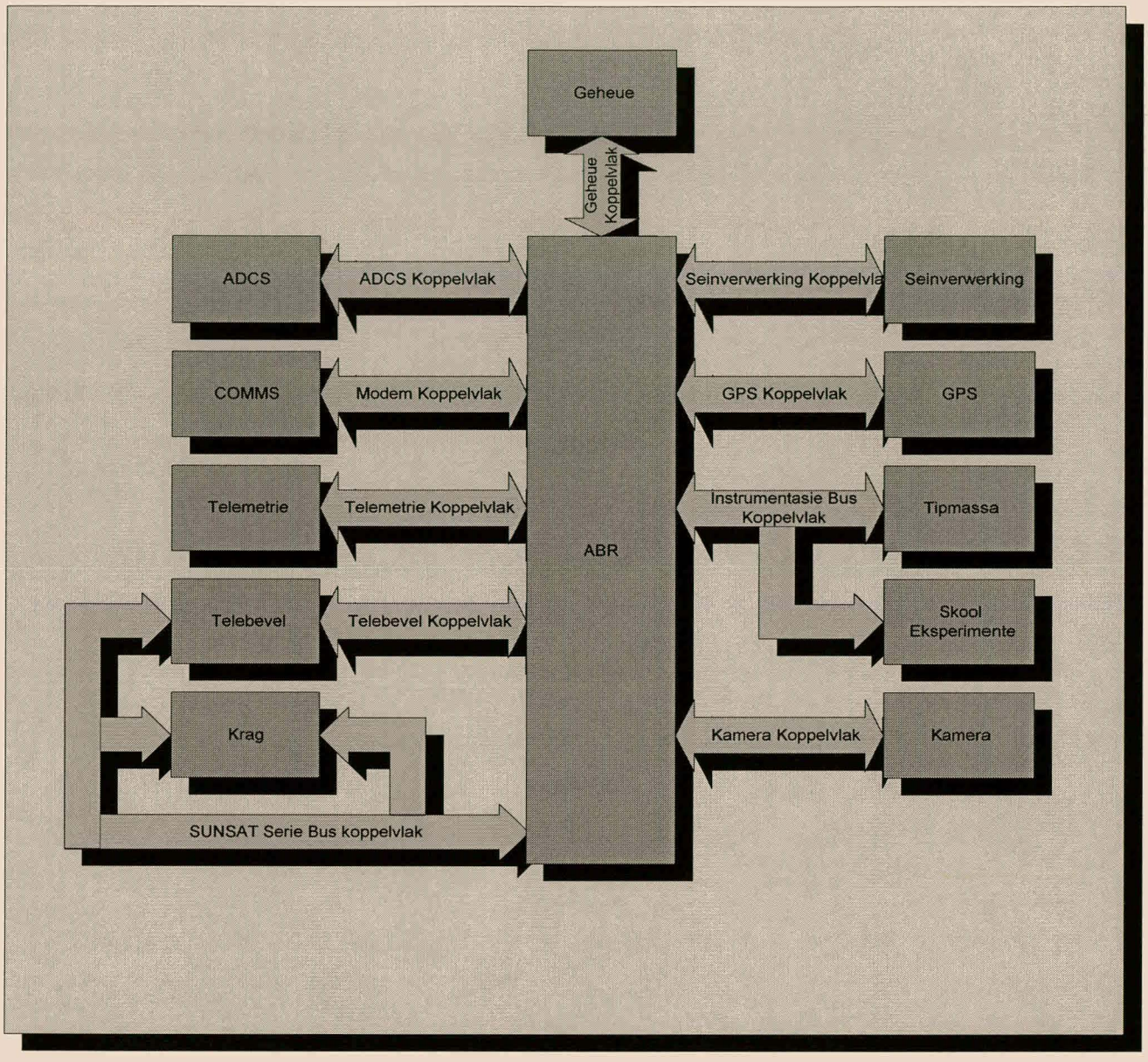
ABR	Aanboordrekenaar
ADCS	"Attitude Determination and Control System"
CAN	"Controller Area Network"
CSMA/CD	"Carrier Sense Multiple Access/Collision Detection with Non Destructive Arbitration"
EGO	Enkel Gebeurtenis Ontwrigting
EIEU	Eerste In Eerste Uit
EMI	Elektromagnetiese Inmenging
FPGA	"Field Programmable Gate Array"
ISA	"Industry Standard Architecture"
ISO/OSI	"International Standardisation Orginisation/ Open System Interconnect"
NRZ	"Non Return to Zero"
SOT	Siklusse Oortolligheids Toets
VHDL	"Very High Speed Integrated Circuit (VHSIC) Hardware Description Language"

Hoofstuk 1: INLEIDING

Die uitleg van die huidige SUNSAT 1 mikro satelliet word in die onderstaande figuur getoon. Die satelliet het 'n modulêre struktuur en bestaan uit aparte modules wat aan mekaar verbind word om die satelliet te vorm.

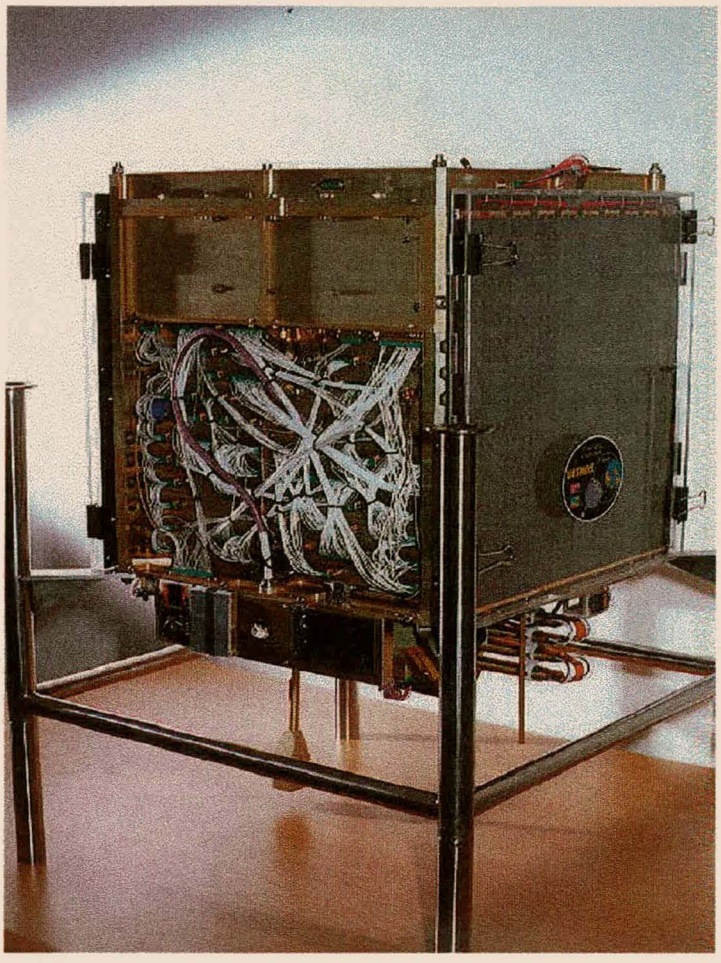
Figuur 1: Modulêre struktuur van SUNSAT 1

Interstelsel kommunikasie geskied deur middel van 'n parallelle argitektuur wat bestaan uit gededikeerde verbindings tussen die aanboordrekenaar (ABR) en elk van die verskillende substelsels op die satelliet. Elke verbinding bestaan uit 'n unieke koppelvlak asook 'n unieke stel drade. 'n Skematiese diagram van die ABR gekoppel aan die verskillende stelsels word in figuur 2 getoon.



Figuur 2: ABR Koppelvlakke

Figuur 3 toon die satelliet met die wye verskeidenheid parallelle interverbindings.



Figuur 3: SUNSAT 1 Draad Harnas

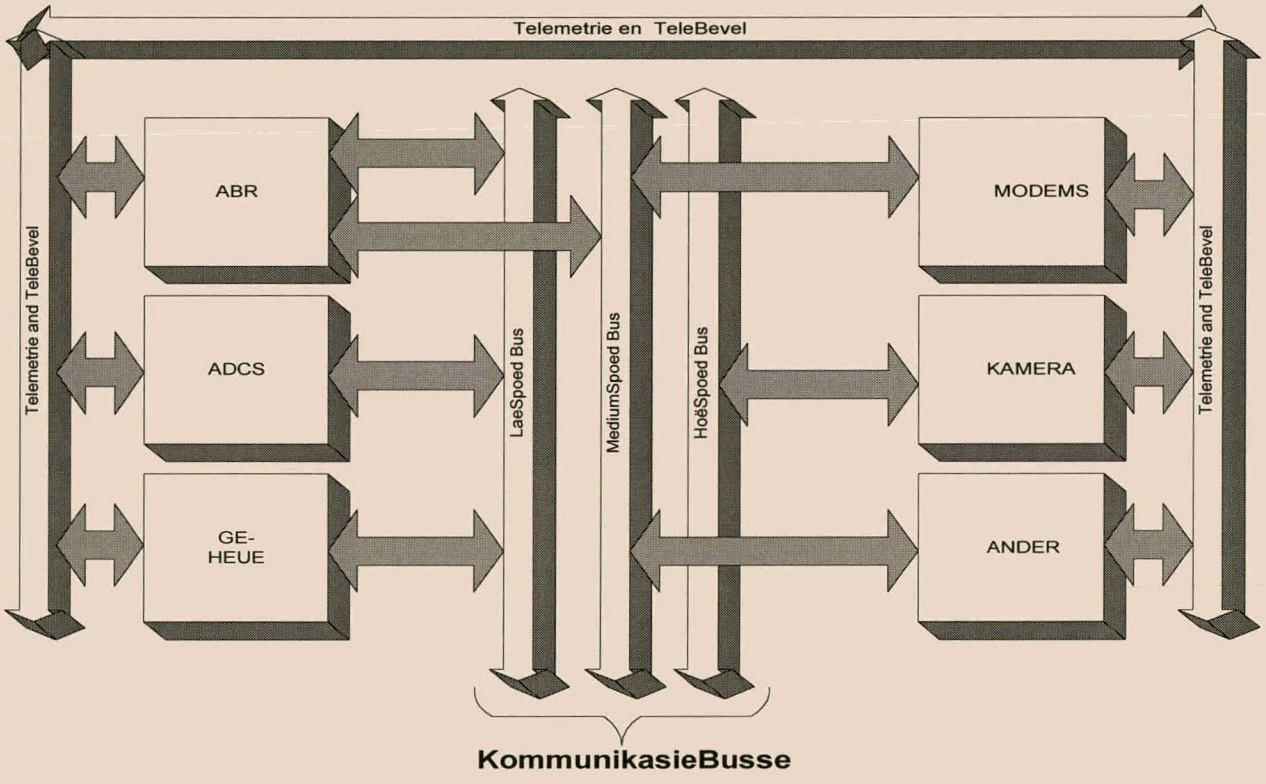
1.1 Probleme met huidige stelsel:

Uit figuur 3 kan gesien word dat die draad harnas wat betrokke is by die verbinding van die verskillende substelsels op die satelliet ekstensief is. Elke substelsel het ook 'n unieke koppelvlak met die ABR. Hierdie eienskap bemoeilik die uitbreibaarheid van die stelsel, aangesien daar 'n nuwe koppelvlak ontwikkel moet word vir elke nuwe stelsel wat in die satelliet geplaas word.

'n Ander ongewenste eienskap van die huidige opstelling op SUNSAT 1 is die hoë onderbrekingslas op die ABR. Dit ontstaan as gevolg van die feit dat daar 'n wye verskeidenheid koppelvlakke bestaan wat deur die ABR gediens moet word. Sodra 'n stelsel data aan die ABR wil oordra, genereer dit 'n onderbreking en die ABR moet dan reageer op die onderbreking. 'n Verdere las op die substelsel sowel as die ABR se verwerkers is die kommunikasie protokol hantering. Die werkverrigting van die substelsel of ABR kan aansienlik afneem indien die protokol hantering en die diens van die onderbrekings te veel verwerker tyd in beslag neem.

1.2 Eienskappe van volgende generasie Kommunikasiestelsel:

Vir 'n volgende generasie SUNSAT mikro satelliet is die behoefte daar gestel om die parallelle interverbindings te vervang met 'n netwerkstelsel. Enige stelsel op die satelliet kan dan op die netwerkstelsel inskakel en met enige ander stelsel op die satelliet kommunikeer. Om aan 'n wye verskeidenheid stelsels se behoeftes te voldoen kan die netwerkstelsel uit byvoorbeeld drie aparte netwerke bestaan, naamlik 'n lae-, medium- en hoëspoed netwerk. Voordele van so 'n stelsel is dat die parallelle interverbindings aansienlik verminder word weens die feit dat daar van 'n netwerkstelsel gebruik gemaak word. 'n Voorstelling van 'n moontlike stelsel word in figuur 4 getoon.



Figuur 4: Voorgestelde struktuur vir volgende generasie satelliet

Aangesien die CAN ("Controller Area Network") protokol stelsel reeds ondersoek word vir implementasie in die Telemetrie en Telebevel stelsel sowel as die ADCS ("Attitude Determination and Control System") het die behoefte ontstaan om die moontlikheid te ondersoek om die CAN stelsel ook te gebruik in die implementasie van die laespoed kommunikasiebus.

Drie verskillende opstellings kan identifiseer word waardeur die CAN protokol in 'n laespoed algemene kommunikasiestelsel gebruik word:

Stelsel 1:

Die netwerk protokol word nie deur die substelsel of ABR hanteer nie, maar deur 'n netwerkeenheid waarmee die substelsel of ABR skakel. 'n Generiese eenheid bestaan dus waarmee verskillende substelsels of die ABR toegang tot die netwerk kan verkry. Alle retransmissies en netwerk protokol word deur die generiese eenheid behartig.

Stelsel 2:

Die substelsel of ABR koppel aan 'n netwerkbeheerder eenheid. Hierdie eenheid behartig die transmissie protokol, maar die opstel van die eenheid en fouthantering moet deur die substelsel of ABR hanteer word.

Stelsel 3:

Die totale netwerk transmissieprotokol word fisies op die verwerker van die substelsel of die ABR hanteer. Al die stelsels wat toegang tot die netwerk verlang moet dus die vermoë hê om die netwerk protokol en die fisiese transmissie te hanteer. Die verwerker benodig ook 'n ingeboude netwerkeenheid vir kommunikasie op die netwerk.

Die ontwikkeling van so 'n stelsel na aanleiding van die 3 bogenoemde moontlike implementasies word verder in hierdie tesis ondersoek. Die doel is om te bepaal op watter manier die CAN protokol die mees optimaal gebruik kan word in 'n algemene laespoed netwerk vir 'n volgende generasie SUNSAT mikro satelliet.

1.3 Waarom word CAN enigsins oorweeg vir 'n data netwerk ?

Die hoof toepassing van CAN is in beheerstelsels waar die grepe wat per pakkie versend kan word klein is. Die maksimum datagrepe per raam vir die CAN protokol is 8 grepe en die maksimum datatempo is 1 Mbisse/Sek. Die CAN protokol word oorhoofs in bylae 1 bespreek. Indien groot hoeveelhede data versend word, moet die data in 'n hele aantal pakkies met 8 datagrepe per pakkie opgebreek word. Die resultaat is dat die effektiewe datatempo aansienlik afneem a.g.v die transmissie van oorkoepelende protokol inligting in elke raam. Die raamgrootte, met bis inplasing geïgnoreer, vir die standaard CAN formaat, met 8 datagrepe per raam, is 111 bisse. Die effektiewe tempo van data oordrag is dus 1Mbit/Sek*(64/111)=576.577 Kbisse/Sek. Aangesien die stelsel ontwerp word vir laespoed kommunikasie is die datatempo van 576.577 Kbisse/Sek aanvaarbaar.

Die klein hoeveelheid data per raam, 8 grepe, veroorsaak ook 'n hoë onderbrekingslas op die verwerker wat gekoppel is aan die CAN eenheid. Indien groot hoeveelhede data van 1 nodus na 'n ander oorgedra word, moet die boodskap in 'n hele aantal kleiner pakkies van 8 datagrepe per pakkie opgebreek word. Na die ontvangs van elk van hierdie kleiner pakkies moet die CAN eenheid die stelsel wat daaraan gekoppel is onderbreek om die pakkie oor te dra. Dit veroorsaak 'n hoër onderbrekingslas in vergelyking met 'n stelsel waar daar meer datagrepe per pakkie versend kan word.

Hierdie eienskap is egter nie van so 'n aard dat dit die gebruik van CAN in data kommunikasie onmoontlik maak nie. Die hoër onderbrekingslas wat ontstaan as gevolg van die klein pakkie grootte moet egter in ag geneem word by die ontwerp en evaluering van die stelsel.

1.4 Uitleg van res van tesis:

Die eerste afdeling beskryf die ontwerp en werking van die 3 bogenoemde stelsels. In die tweede afdeling word die eienskappe van die 3 stelsels met mekaar vergelyk. 'n Gevolgtrekking word ook gemaak en verdere werk wat gedoen moet word om 'n nuwe generasie kommunikasiestelsel daar te stel word ook bespreek.

Afdeling 1: Ontwerp en beskrywing van stelsels

- ✓ Hoofstuk 2: Beskrywing van stelsel 1
- ✓ Hoofstuk 3: Detail Ontwerp van stelsel 1
- ✓ Hoofstuk 4: Ontwerp en beskrywing van stelsel 2
- ✓ Hoofstuk 5: Ontwerp en beskrywing van stelsel 3

Hoofstuk 2: Stelsel 1 Beskrywing

2.1 Agtergrond:

Een van die behoeftes wat daar gestel is met die ontwerp van 'n volgende generasie kommunikasiestelsel is dat die onderbrekingslas op die substelsel verwerker of die ABR so laag as moontlik gehou moet word. Soos in die inleiding bespreek is die CAN protokol in hierdie opsig nie ideaal nie. Die netwerkstelsel wat aan die substelsel of ABR koppel kan egter so ontwerp word dat dit die opbreek van boodskappe wat versend word en die heropbou van boodskappe ontvang oor die netwerk self hanteer. In so 'n opstelling is die onderbrekingslas op die substelsel of die ABR heelwat laer aangesien die substelsel of ABR slegs onderbreek word indien 'n volledige boodskap en nie aparte rame, ontvang is. Die feit dat slegs 8 datagrepe per raam versend kan word, is dus nie sigbaar vir die substelsel of ABR wat die laespoed netwerk gebruik nie.

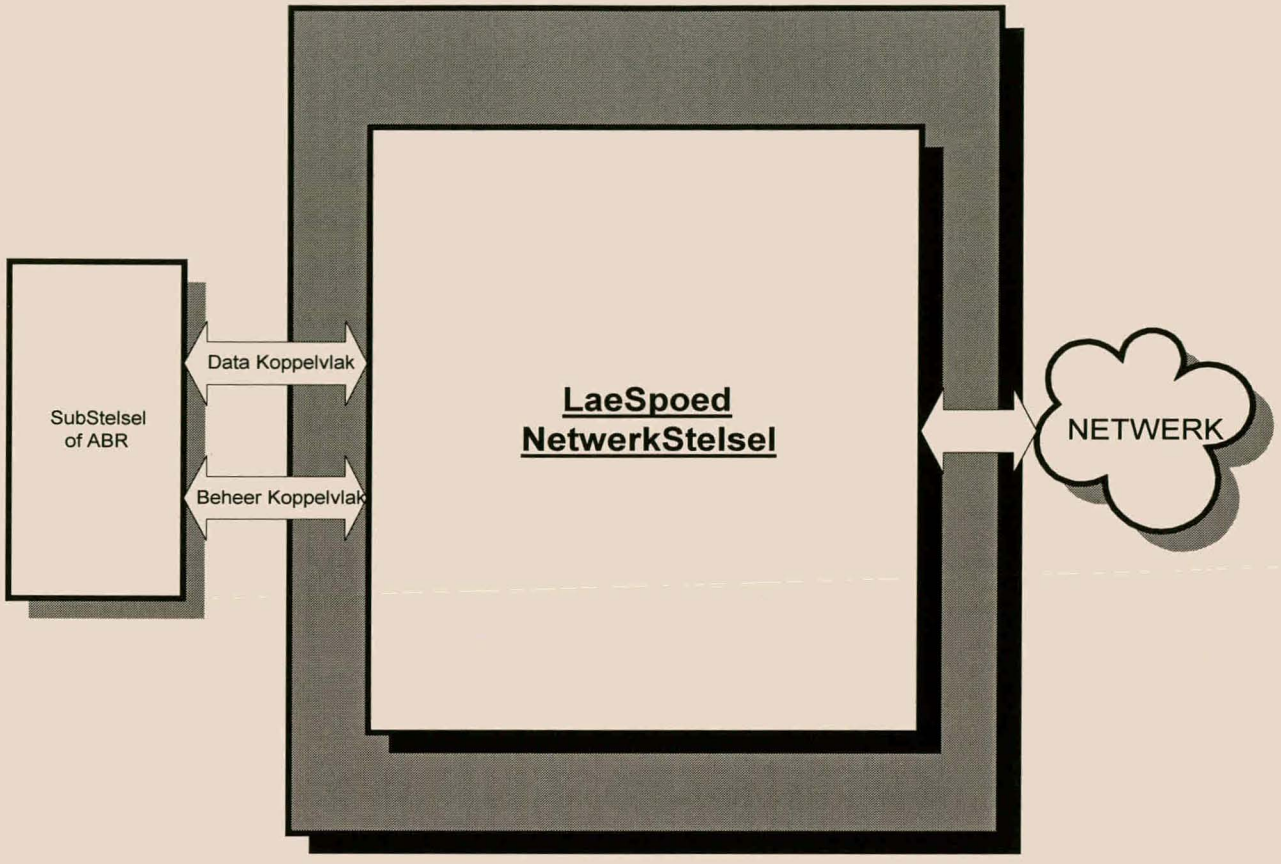
2.2 Werking:

Die substelsels verkry toegang tot die laespoed kommunikasiebus deur middel van 'n netwerkkoppelvlak eenheid.

Die netwerkkoppelvlak eenheid is verantwoordelik vir adressering en transmissie van boodskappe, foutdeteksie, retransmissies, hantering van netwerk faling en netwerktoegang beheer. Weens die feit dat die netwerkstelsel met 'n verskeidenheid van stelsel geïntegreer moet kan word, moet die grootste gedeelte van die verwerkerlas wat met versending en ontvangs van boodskappe oor 'n netwerk gepaard gaan, op die netwerkstelsel rus en nie op die substelsel wat aan die netwerksel gekoppel is nie.

Die opstelling van die substelsel of die ABR gekoppel aan die laespoed netwerkstelsel word in figuur 5 getoon.

Netwerk Koppelvlak Eenheid:



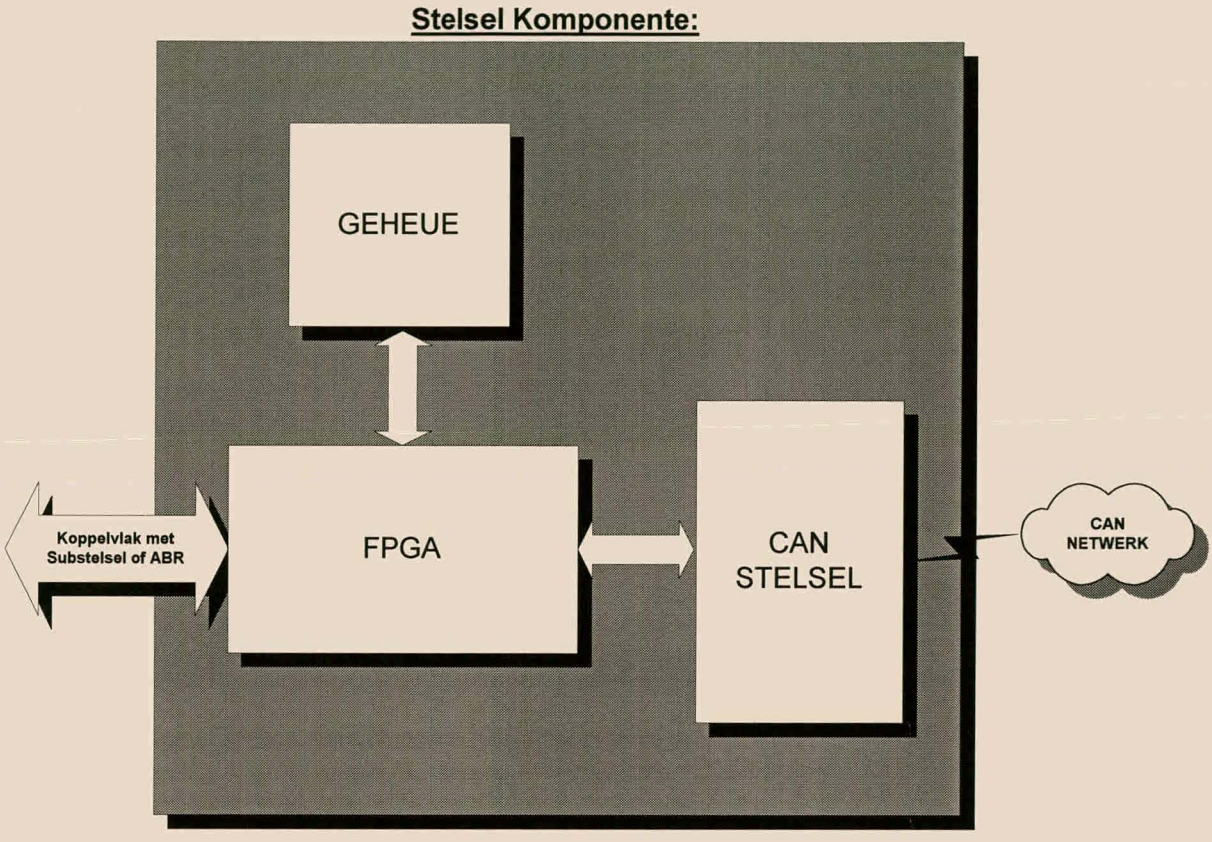
Figuur 5: Netwerkkoppelvlak eenheid

Die tipiese werking van die stelsel is as volg:

Die toepassing stel die netwerkstelsel deur middel van die beheer koppelvlak in kennis dat dit ‘n boodskap van ‘n sekere lengte aan ‘n ander toepassing, met ‘n sekere adres, wil stuur. Die netwerkstelsel ontvang dan die hele boodskap vanaf die toepassing, greep vir greep, deur middel van die data koppelvlak. Die netwerkstelsel breek dan, indien nodig, die boodskap in kleiner rame op en versend die boodskap raam vir raam oor die netwerk na die gekose destinasie. Die netwerkstelsel waarna die data gestuur word ontvang die data en stoor dit tydelik totdat die hele boodskap ontvang is. Sodra die hele boodskap ontvang is, stel daardie netwerkstelsel die substelsel daaraan gekoppel in kennis dat daar ‘n boodskap is wat gelees moet word en die substelsel kan dan die boodskap vanaf die netwerkeenheid lees.

2.3 Komponente van Laespoed Netwerkstelsel:

Die verskillende komponente van die netwerkstelsel word in onderstaande figuur getoon.



Figuur 6: Komponente van Laespoed Netwerkstelsel

2.3.1 FPGA ("Field Programmable Gate Array"):

- Die FPGA dra die verantwoordelikheid om die koppelvlak tussen die substelsel of ABR en die netwerkstelsel te hanteer.
- Die FPGA is verantwoordelikheid vir die identifisering van geheue bis foute.
- Geheue bestuur word deur die FPGA behartig.
- Daar is ook verskeie ander take soos die opwek van onderbrekings wat deur die FPGA administreer word.

2.3.2 Geheue:

Die geheue in die stelsel dien as tydelike stoorplek vir boodskappe wat oor die netwerk gestuur of ontvang word. Die doel van die geheue is om die verwerkerlas wat betrokke is by die ontvang van afsonderlike grepe van 'n sekere boodskap te verlig. Soos wat die afsonderlike rame van 'n sekere boodskap oor die netwerk ontvang word, stoor die netwerkstelsel die boodskap tydelik in die geheue totdat die volledige boodskap ontvang is. Sodra die boodskap dan volledig ontvang is, stel die netwerkstelsel die substelsel wat daaraan gekoppel is in kennis en die substelsel kan die volledige boodskap uit die geheue lees.

Die geheue stel ook die stelsel in staat om meer as een boodskap op 'n slag te stoor vir versending, sowel as vir ontvangs. Dus terwyl die netwerkeenheid besig is om een boodskap uit die geheue te versend kan die substelsel of ABR 'n nuwe boodskap op 'n ander posisie in die geheue stoor.

2.3.3 CAN Stelsel:

Die CAN stelsel word benodig vir die implementering van die netwerk protokol en die fisiese transmissie van data oor die netwerk. Die CAN stelsel skakel met die FPGA vir data oordrag. Dit is ook verantwoordelik vir die versending en ontvangs van data oor die netwerk.

2.4 Hardeware Opstelling:

Om die stelsel te toets is 'n kaart ontwikkel wat die bostaande komponente bevat. Die koppeling met 'n substelsel of 'n ABR is vir toets doeleindes vervang met die standaard persoonlike rekenaar ISA bus koppelvlak. Dit is gedoen aangesien die eienskappe van die persoonlike rekenaar se ISA bus en die eienskappe van die koppelvlakke van die substelsels op die satelliet of die ABR nou ooreenstem. Dit is ook maklik om die stelsel werking te toets met die persoonlike rekenaar. Die stroombaandiagram van die kaart word in bylae 9 aangegee.

Hoofstuk 3: Stelsel 1 Detail Ontwerp

3.1 FPGA:

3.1.1 Keuse van FPGA:

Die ontwikkeling van die logika wat op die FPGA implementeer is, is in VHDL ("Very High Speed Integrated Circuit (VHSIC) Hardware Description Language") gedoen. VHDL is 'n standaard wat deur alle FPGA vervaardigers ondersteun word en dit is relatief maklik om die FPGA vervaardiger te verander deur net die VHDL vir die toepassing te herkompileer vir die nuwe FPGA.

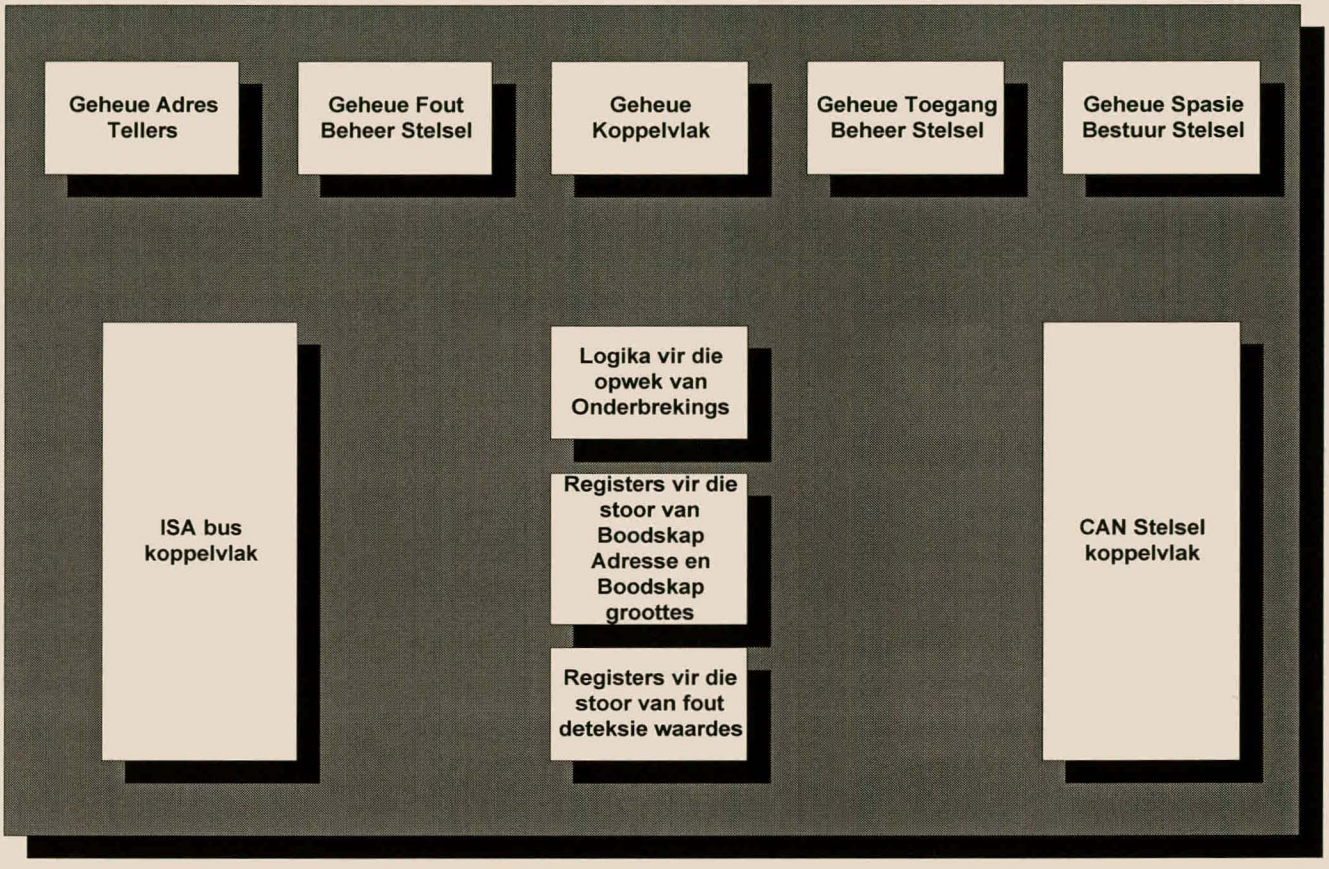
Vir enige ruimte toepassing is dit belangrik om komponente te gebruik wat radiasiebestand is en wat effektief sal funksioneer in 'n ruimte omgewing. Daar bestaan ook FPGA's wat spesifiek vir ruimte toepassings vervaardig word. Hierdie FPGA's maak van "fusible link" tegnologie gebruik. Die enigste nadeel van hierdie tipe komponente is dat dit nie herprogrameerbaar is nie. Hierdie feit bemoeilik die ontwikkeling en uitbreiding van die stelsel. As gevolg hiervan is die keuse gemaak om die ontwikkeling van die stelsel op 'n ALTERA FPGA te doen. Die ALTERA FPGA's maak van statiese geheue tegnologie gebruik en kan maklik herkonfigureer word. Die eienskappe van die stelsel met die ALTERA FPGA sal genoegsame inligting verskaf om die werkverrigting van die radiasiebestande FPGA stelsel af te skat. Indien so 'n stelsel dan op 'n volgende generasie satelliet geïmplementeer word, kan daar net oorbeweeg word na die radiasiebestande tegnologie.

Die oorwegings wat in ag geneem is by die keuse van die FPGA is grootte, spoed, koste en vertroudheid met die ontwikkelings omgewing. Die FPGA wat gekies is, is die ALTERA EPF6016QC208-3 met 1320 logiese elemente. Hierdie FPGA kos ongeveer R180.00.

3.1.2 Substelsels op FPGA geïmplementeer:

Verskillende substelsels word op die FPGA geïmplementeer wat 'n verskeidenheid funksies verrig. 'n Diagram van die verskillende substelsels teenwoordig op die FPGA word in figuur 7 getoon.

FPGA Substelsels:



Figuur 7: FPGA Substelsels

Die belangrikste stelsels word in die volgende subafdelings bespreek. Die VHDL kode vir die implementasie van die verskillende stelsels word in bylae 10 aangegee.

3.1.2.1 Koppelvlakke:

Die FPGA is verantwoordelik vir die bestuur van twee koppelvlakke naamlik die koppelvlak met die substelsel of ABR en die CAN stelsel. Beide koppelvlakke word gebruik vir bidireksionele oordrag van boodskappe sowel as algemene informasie oordrag.

Vereistes vir koppelvlakke:

- Data moet in beide rigtings oor die koppelvlak gestuur word.
- Die substelsel, ABR of CAN stelsel moet in beheer van die data oordrag na en van die FPGA wees.
- Die FPGA moet die substelsel, ABR of CAN stelsel in kennis kan stel van sekere veranderinge.
- Die datapoort moet 8 bisse wyd wees om versoenbaar te wees met 'n verskeidenheid verwerkers.

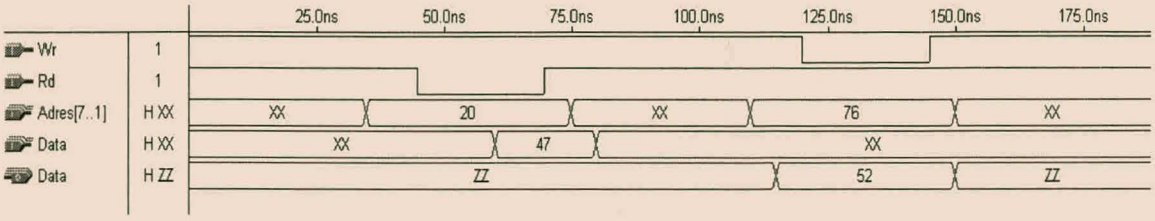
Ontwerp:

Die stelsel wat ontwikkel is om aan bogenoemde vereistes te voldoen bestaan uit 'n afsonderlike adres- en databus met 'n asinchrone lees- en skryfsiklus. Die rede hiervoor is dat meeste verwerkers in staat is om eksterne asinchrone lees- en skryfsiklusse uit te voer. Met 'n asinchrone lees- en skryfstelsel is dit maklik om boodskappe van verskillende betekenis oor te dra tussen die twee stelsels. Dit word gedoen deur 'n sekere betekenis te heg aan data wat na 'n spesifieke adres geskryf word. Byvoorbeeld, indien die substelsel, ABR of CAN stelsel die grootte van 'n boodskap aan die FPGA wil oordra kan dit 'n skryfsiklus uitvoer by adres 1. Om die eindbestemming van 'n boodskap aan te dui kan die substelsel, ABR of CAN stelsel byvoorbeeld 'n skryfsiklus by adres 2 uitvoer en om die data oor te dra kan daar herhaaldelik na adres 3 geskryf word. Die substelsel, ABR of CAN stelsel voer dus sekere lees- en skryfinstruksies by sekere adresse uit en die FPGA reageer dan op die instruksies. Die FPGA kan die substelsel, ABR of CAN stelsel in kennis stel van sekere veranderings deur middel van onderbrekingslyne.

Koppelvlak Lyne tussen netwerk nodus en substelsel:

1. Data(7..0)
2. Adres(7..0)
3. Rd
4. Wr
5. Int

'n Lees en skryfsiklus word in onderstaande figuur getoon.

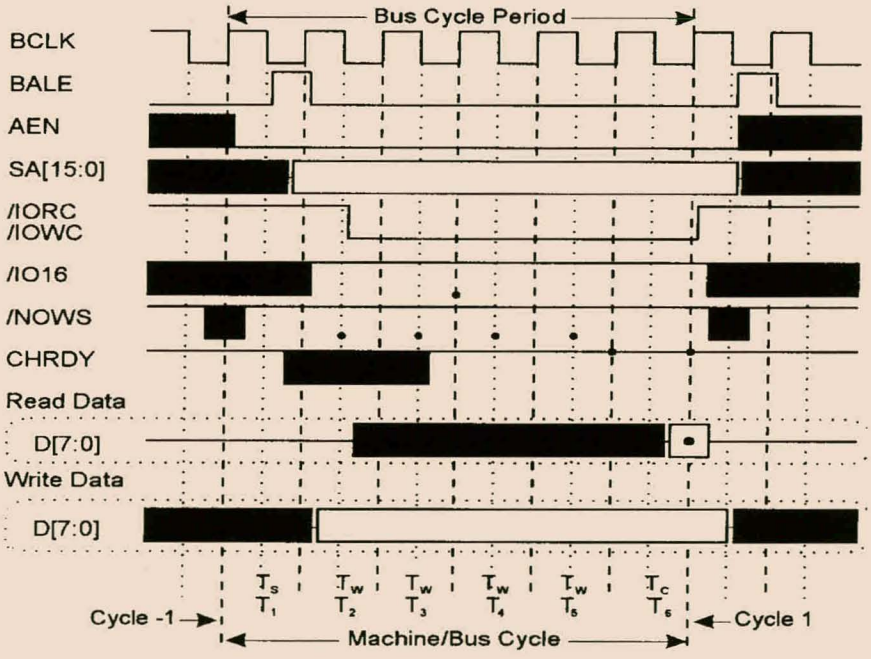


Figuur 8: Asinchrone lees en skryfsiklus

Implementasie:

1. ISA Bus Koppelvlak:

Die ISA bus word in die standaard 6 busklok, 8 bis ingang/uitgang modus gebruik. 'n Tyd diagram van die ingang/uitgang toegang siklus word in figuur 9 getoon.



Figuur 9: ISA Bus lees en skryfsiklusse

Die adres bus word ingelees op die afgaande rand van die BALE sein ingelees slegs indien die AEN sein laag is. Indien die adres binne die toegekende adres reeks van die FPGA (300H – 310H) val reageer die FPGA op die lees- en skryfaksies deur die persoonlike rekenaar uitgevoer. Die klokspoed van die ISA bus is tussen 10 MHz en 12 MHz. Indien daar 'n skryfaksie uitgevoer word deur die persoonlike rekenaar, word die data op die opgaande flank van die IOWC lyn ingegrendel en indien die persoonlike rekenaar data vanaf die FPGA wil lees, word die data deur die FPGA op die databus geskryf solank IORC laag is.

2. CAN Stelsel Koppelvlak:

Kommunikasie tussen die FPGA en die CAN verwerker vind plaas deurdat die FPGA in die eksterne adres spasie van die CAN verwerker gekarteer is. Net soos die geval met die ISA bus stelsel reageer die FPGA ook op eksterne lees en skryfsiklusse van die CAN verwerker, indien die adres van eksterne toegang van die CAN verwerker binne die gekarteerde adres spasie van die FPGA val.

3.1.2.2 Geheue Fout Beheer:

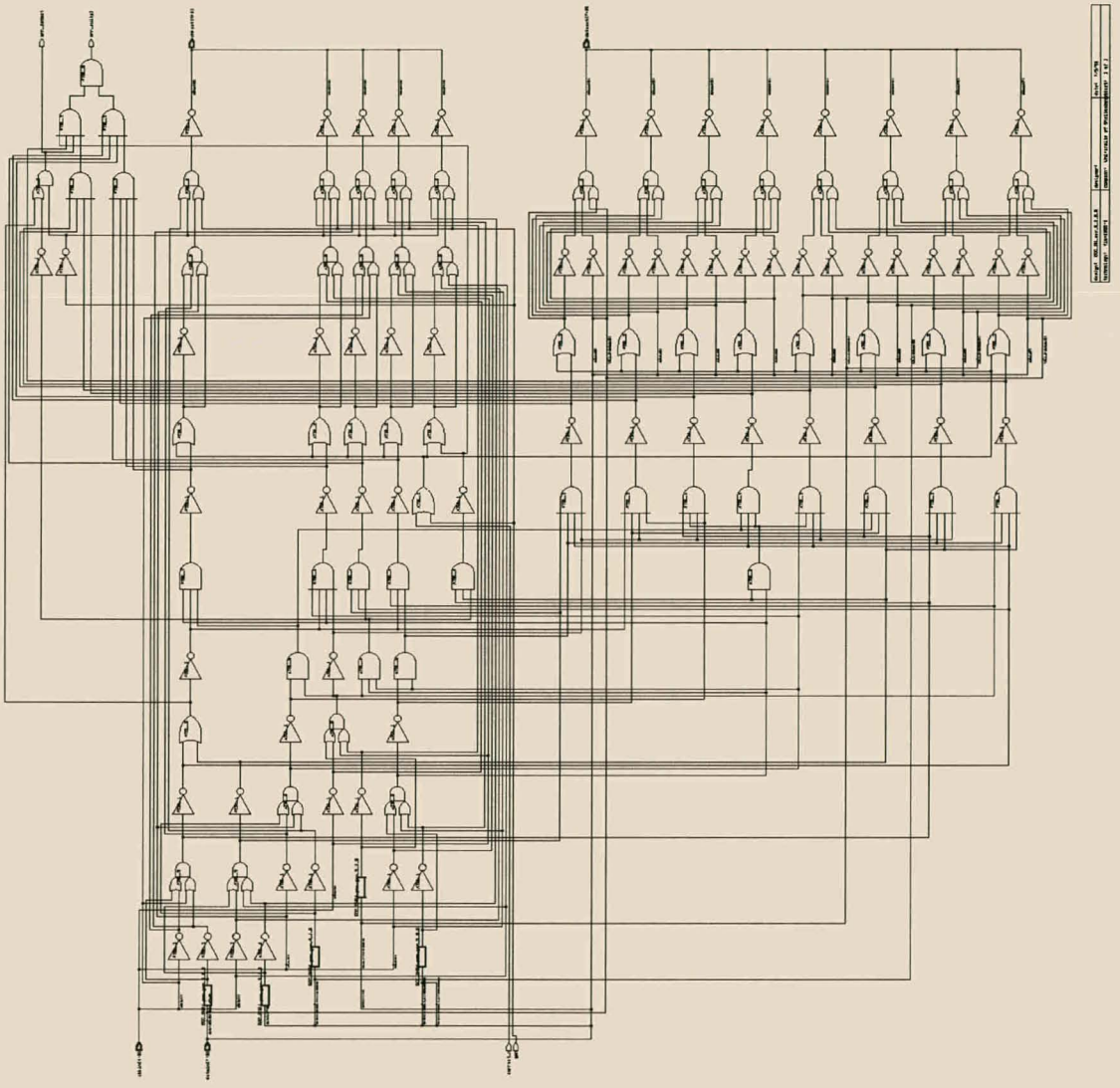
Partikels is subatomiese deeltjies soos protone, elektrone, neutrone, alfa-partikels en beta-partikels. Hierdie partikels kom in die ruimte met verskillende energie en digthede voor. Gelaaide partikels benadeel die satelliet se elektronika op hoofsaaklik twee maniere:

1. Dit verhoog elektrostatiese lading in die satelliet.
2. Dit veroorsaak die voorkoms van "enkel gebeurtenis ontwrigting" (EGO).

'n EGO vind plaas wanneer 'n gelaaide partikel die satelliet binnedring en die korrekte werking van die elektronika versteur. Sodra so 'n partikel met hoë energie die satelliet binnedring kan dit veroorsaak dat 'n grendel of 'n geheue element in die elektronika op die satelliet se waarde verander word. Dit staan bekend as 'n EGO. Gewoonlik beskadig 'n EGO nie die elektronika waarin dit plaasgevind het nie en die enigste uitwerking wat dit het is dat die waarde van die grendel of geheue element verander.

Weens die feit dat die boodskappe vanaf die substelsel, ABR en CAN stelsel eers in die geheue gestoor word, moet daar 'n meganisme geïmplementeer word om die integriteit van die data te verseker deur EGO's te identifiseer.

Een metode om hierdie EGO's te identifiseer en in sekere gevalle te korrigeer is om van 'n foutdeteksie en korreksie stelsel gebruik te maak. Die werking van so 'n stelsel word in bylae 2 bespreek. 'n Foutdeteksie en korreksie stelsel is geïmplementeer met behulp van SYNOPSYS se DESIGNWARE komponente. Die skematiese diagram van die stelsel, wat met behulp van SYNOPSYS se DESIGN ANALYZER verkry is, word in die onderstaande figuur getoon.



Figuur 10: Skematiese Diagram van Foutdeteksie en korreksie

Hierdie foutdeteksie en korreksie stelsel gebruik 55 logiese selle in die EPF6016QC208-3 ALTERA FPGA, wat 'n relatief klein persentasie (4 %) van die totale aantal logiese selle (1320) in hierdie FPGA is. Die geskatte stroomverbruik vir bogenoemde implementasie, volgens formules deur Altera verskaf, is 12.5 mA. Die

berekening van hierdie waarde word in bylae 3 getoon. Hierdie stelsel neem die 8 bis ingang en genereer 'n 5 bis toets reeks vir daardie ingang. Hierdie 12 bisse word dan in geheue gestoor en wanneer die data uit die geheue gelees word, bepaal die stelsel of die of die data gekorrupteer is m.b.v. die ekstra 5 bisse. Alle enkel bis foute word gekorrigeer sowel as sommige dubbel bis foute. Die nadeel van die stelsel is dat die data wydte van 8 bisse na 12 bisse verhoog. Om dieselfde hoeveelheid inligting met een skryf of leessiklus te stoor moet die datapoort wydte verhoog word vanaf 8 na 12 bisse. Dit het die gevolg dat daar van geheue met 'n datapoort van ten minste 12 bisse gebruik gemaak moet word. Aangesien statiese geheue met 'n ingang data wydte van 12 bisse nie maklik beskikbaar is nie, moet daar van 16 bis geheue gebruik gemaak word om so 'n stelsel te implementeer. In die onderstaande tabel word die eienskappe van die 16 en 8 bis asinchrone statiese geheue met mekaar vergelyk.

Totale Grootte:	Uitleg:	Operasionele Stroomverbruik:	Bystands Stroomverbruik:
1 Meg	64k X 16	160 mA	55 mA
1 Meg	128k X 8	120 mA	55 mA

Tabel 1: Geheue Stroomverbruik

Daar kan dubbel soveel inligting in 'n 1 Meg 128k X 8 blok geheue gestoor word sonder enige foutdeteksie en korreksie stelsel as in 1 Meg 64k X 16 geheue met 'n foutdeteksie en korreksie stelsel. Dit rede hiervoor is dat 3 van die sestien bisse nie gebruik word nie en 'n verdere 5 word gebruik om die toets sekwensie van die foutdeteksie en korreksie stelsel te stoor.

Indien die stroomverbruik dus vergelyk word moet die stroom waardes van die 1 Meg 64k X 16 geheue dus verdubbel word voordat dit met die 1 Meg 128k X 8 bis geheue vergelyk kan word. In die geval van die operasionele stroomverbruik gebruik die stelsel met die foutdeteksie en korreksie eenheid en die 1 Meg 64k X 16 geheue dus meer as dubbel die hoeveelheid stroom as in die geval van die 8 bis geheue. Die bystands stroom in die geval van die 16 bis geheue is dubbel die stroom van die 8 bis geval.

Die toename in stroomverbruik om 1 Meg se boodskappe met behulp van die foutdeteksie en korreksie stelsel te stoor in vergelyking met 'n stelsel daar sonder word in onderstaande tabel getoon. In die tabel word die stroomverbruik van die ekstra register in die FPGA wat die toets sekwensie stoor en die ekstra 5 I/O penne wat aan die geheue verbind is as weglaatbaar klein aanvaar.

Beskrywing:	Toename in Stroom:
Geheue Operasionele Stroom	200 mA
Foutdeteksie en Korreksie implementasie in die FPGA	12.26 mA
Totaal:	212.26 mA

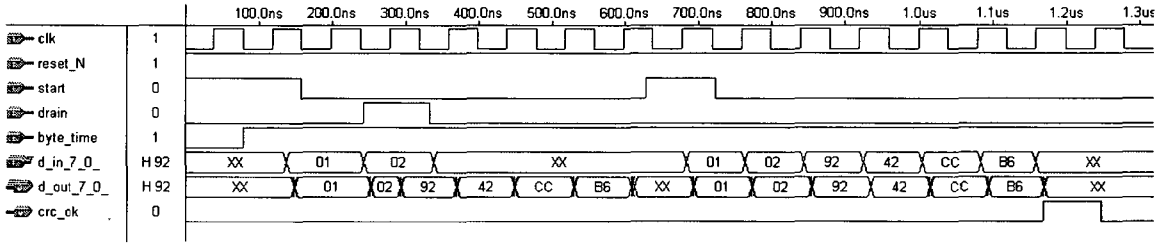
Tabel 2: Opsomming van Foutdeteksie en Korreksie stelsel stroomverbruik

'n Tweede metode om 'n EGO op te spoor is om gebruik te maak van SOT (Siklusse Oortolligheids Toets). Die werking van SOT word in bylae 4 bespreek. In hierdie opstelling word daar van geheue met 'n 8 bis datapoort gebruik gemaak. Soos wat die data deur die toepassing in die geheue geskryf word, word daar 'n SOT toets som van 32 bisse bereken. Hierdie waarde word dus bereken voordat die data in die geheue gestoor word en beïnvloed kan word deur een of meer EGO's. Die waarde word dan saam met die boodskap oor die netwerk gestuur. By die ander stasie word die boodskap in die geheue gestoor, totdat daardie stelsel se toepassing gereed is om die boodskap uit die geheue te lees. Soos wat die boodskap aan die toepassing gelewer word, bereken die FPGA 'n nuwe SOT waarde met die ontvangde data as ingang. Sodra die hele boodskap aan die toepassing gelewer is, vergelyk die FPGA die ontvangde SOT waarde met die berekende SOT waarde. Indien die 2 ooreenstem het daar onder andere geen EGO's voorgekom nie. 'n Verdere voordeel van die stelsel is dat dit ook verseker dat daar nie enige ander foute voorkom tydens die versending van die data nie.

Die SOT genereerder en toetser is deur middel van SYNOPSYS se DESIGNWARE komponente geskep. Twee ander opsies is ook oorweeg naamlik om die VHDL vir die SOT self te skryf of om een van ALTERA se derde party ontwikkelaars se SOT funksie te gebruik. Die ALTERA derde party funksies moet egter afsonderlik aangekoop word en weens die rede is hierdie opsie nie gekies nie. Aangesien die SYNOPSYS SOT funksie beskikbaar was, is daar gevoel dat dit nie 'n sinvolle besteding van tyd sou wees om die SOT funksie se VHDL kode self te skryf nie en die SYNOPSYS DESIGNWARE SOT funksie is gekies vir evaluasie.

In die onderstaande figuur word die resultate, wat met behulp van die MAXPLUS sagteware verkry is, van die simulasie van die SOT stelsel getoon. Eerstens word daar twee waardes 01 en 02 by die ingang van die stelsel aangelê. Daarna word die heksadesimale 32 bis SOT waarde vir die twee ingang waardes by die uitgang verkry, 8 bisse op 'n keer met die mees belangrikste waarde eerste naamlik 92H, 42H, CCH en B6H. Daarna word daar voortgegaan met die toetsing van die waardes deur eers 01 dan 02 en laastens die SOT (92H, 42H, CCH, B6H) by die ingang poort aan te lê.

Nadat die laaste 8 bisse by die ingang poort aangelê is dui die "Crc_Ok" uitgang aan dat die SOT waarde vir die 2 ingang waardes korrek is.



Figuur 11: SOT Werking

Aangesien daar baie vlakke van logika tussen die uitgang en die ingang is, beperk dit die maksimum frekwensie waarteen die stelsel kan funksioneer. Met simulasies in ALTERA is die maksimum klokfrekwensie wat gebruik kan word bepaal op ongeveer 25MHz. Aangesien die stelsel klokspoed 12 MHz is, word daar aan die stelsel spesifikasies voldoen.

Die SOT stelsel gebruik 119 logiese selle in die EPF6016QC208-3 ALTERA FPGA, wat 9% van die totale aantal logiese selle (1320) in hierdie FPGA is. Die geskatte stroomverbruik vir bogenoemde implementasie volgens formules deur Altera verskaf, is 20.708 mA. Die berekening van hierdie waarde word in bylae 5 getoon.

Vergelyking tussen Foutdeteksie en Korreksie en SOT metodes:

Voordele van Foutdeteksie en Korreksie stelsel bo SOT:

- Die foutdeteksie en korreksie stelsel werk op greep vlak, terwyl die SOT stelsel op boodskap vlak werk.
- Die foutdeteksie en korreksie stelsel is in staat om sekere foute te korrigeer, terwyl die SOT stelsel slegs foute kan identifiseer.
- Met die foutdeteksie en korreksie stelsel word foute opgespoor sodra data uit geheue gelees word waar foutiewe data in die geval van die SOT oor die netwerk versend kan word en die fout eers by die ontvangstelsel opgespoor word. Met die SOT stelsel moet die hele boodskap dan ook weer oor die netwerk versend word.
- Geen ekstra data word oor die netwerk versend soos in die geval van die SOT waar die 4 ekstra SOT grepe oor die netwerk versend moet word.

Voordele van SOT stelsel bo foutdeteksie en korreksie stelsel:

- Die SOT stelsel se stroomverbruik is minder as die foutdeteksie en korreksie stelsel.
- Geheue met 'n 8 bis datapoort kan gebruik word met die SOT stelsel in vergelyking met die 16 bis geheue vir die foutdeteksie en korreksie stelsel.

Keuse van Fout beheer Stelsel;

Op die huidige SUNSAT 1 se ABR met 512K statiese geheue kom daar gemiddeld 3 tot 4 EGO's per week voor. Aangesien hierdie syfer baie laag is, is die SOT stelsel, met 'n laer stroomverbruik in vergelyking met die foutdeteksie en korreksie stelsel, gekies vir implementasie.

Vir toepassings waar EGO's dikwels voorkom en die tyd bestee aan die korrigering van foutiewe data noemenswaardig word, sal die foutdeteksie en korreksie stelsel ten spyte van die hoër stroomverbruik 'n beter opsie wees. Waar EGO's gereeld voorkom is die SOT stelsel oneffektief in die sin dat foute wat reeds in die stuurstelsel se geheue ontstaan eers by die ontvangstelsel opgespoor word. Dit beteken dat die foutiewe boodskap oor die netwerk versend word en onnodige las op die netwerk plaas.

3.1.2.3 Geheue Toegang Bestuur:

Twee stelsels verlang toegang tot die geheue naamlik die substelsel of ABR en die CAN stelsel. Beide hierdie stelsels wil boodskappe in die geheue skryf sowel as boodskappe uit die geheue lees. Aangesien die FPGA in beheer is van die toegang tot die geheue moes daar 'n stelsel geïmplementeer word om die wedersydse toegang tot die geheue te bestuur.

Een moontlikheid is om geheue toegang vir 1 stelsel te weier, terwyl die ander stelsel besig is om data na die geheue te skryf of data uit die geheue te lees. Alhoewel hierdie stelsel relatief maklik is om te implementeer skep dit onnodige wagperiodes waartydens 'n stelsel nie met die geheue kan kommunikeer nie.

'n Tweede moontlikheid is om deur middel van die FPGA 'n dubbel poort geheue, waar beide stelsels gelyktydig met die geheue kan kommunikeer, na te boots. Dit word gedoen deur 'n toestandsmasjien te skep wat die geheue toegang administreer en virtuele gelyktydige toegang deur beide stelsels moontlik maak. So 'n stelsel lei tot meer optimale gebruik van die geheue.

Die tweede opsie is dan ook gekies en die verskillende toestande van die toestandsmasjien, wat die geheue toegang beheer, word in tabel 3 aangetoon.

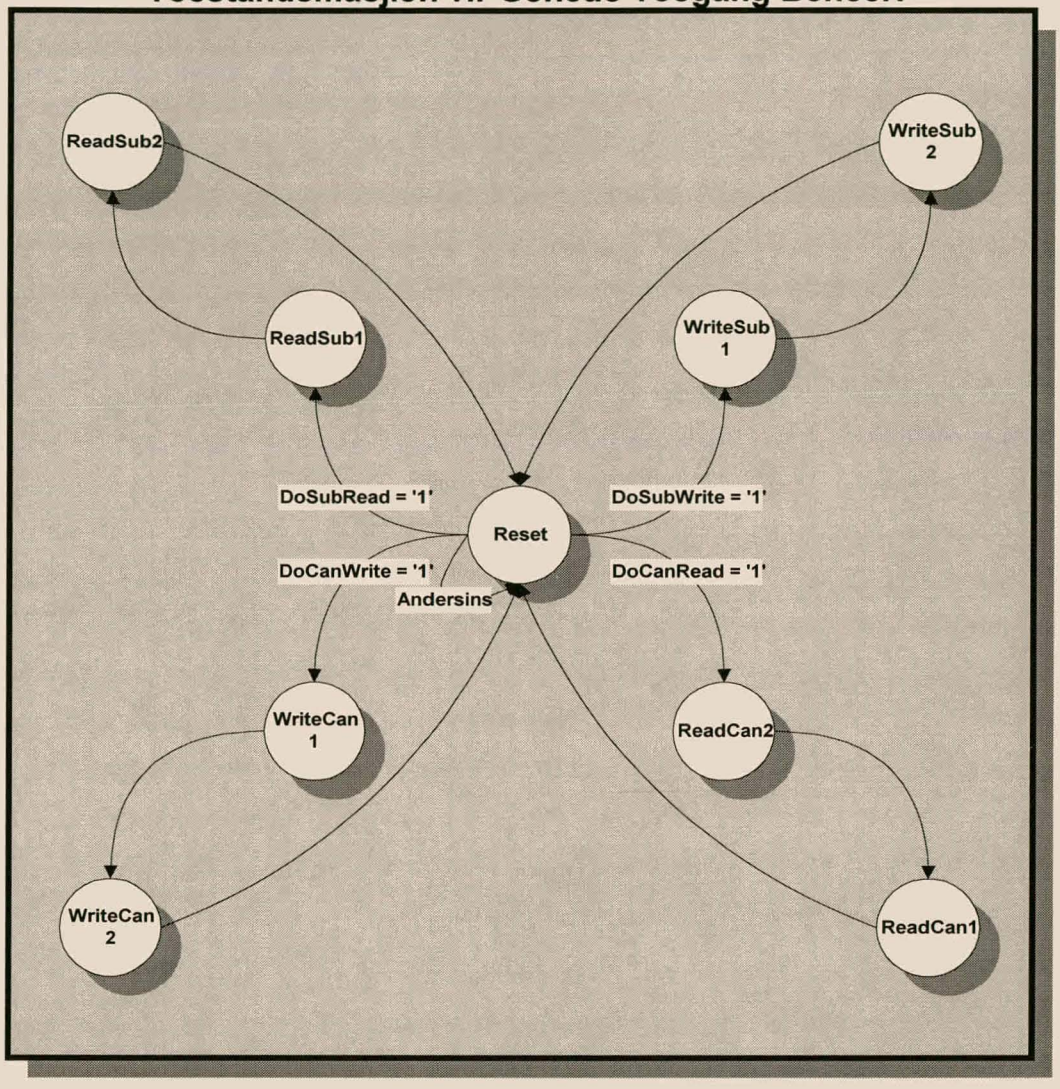
Toestand Nommer:	Toestand Naam:	Betekenis:
0	Reset	Rus Toestand, geen geheue toegang
1	ReadSub1	Eerste stap van substelsel leessiklus
2	ReadSub2	Tweede stap van substelsel leessiklus
3	WriteSub1	Eerste stap van substelsel skryfsiklus
4	WriteSub2	Tweede stap van substelsel skryfsiklus
5	ReadCan1	Eerste stap van CAN stelsel leessiklus
6	ReadCan2	Tweede stap van CAN stelsel leessiklus
7	WriteCan1	Eerste stap van CAN stelsel skryfsiklus
8	WriteCan2	Tweede stap van CAN stelsel skryfsiklus

Tabel 3: Geheue Toestandsmasjien toestande

Die huidige toestand en die waarde van vier vlaggies DoSubRead, DoSubWrite, DoCanRead en DoCanWrite bepaal die volgende toestand van die toestandsmasjien. Die toestande word op elke afgaande rand van die klok opgedateer. Die vier vlaggies

dui aan of die substelsel of die CAN stelsel uit die geheue wil lees of skryf. Die toestandsdiagram word in figuur 12 getoon. Die toestand waarin die toestandsmasjien verkeer bepaal die data, beheer en adres seine wat by die geheue aangelê word.

Toestandsmasjien vir Geheue Toegang Beheer:



Figuur 12: Toestandsdiagram vir geheue toegang

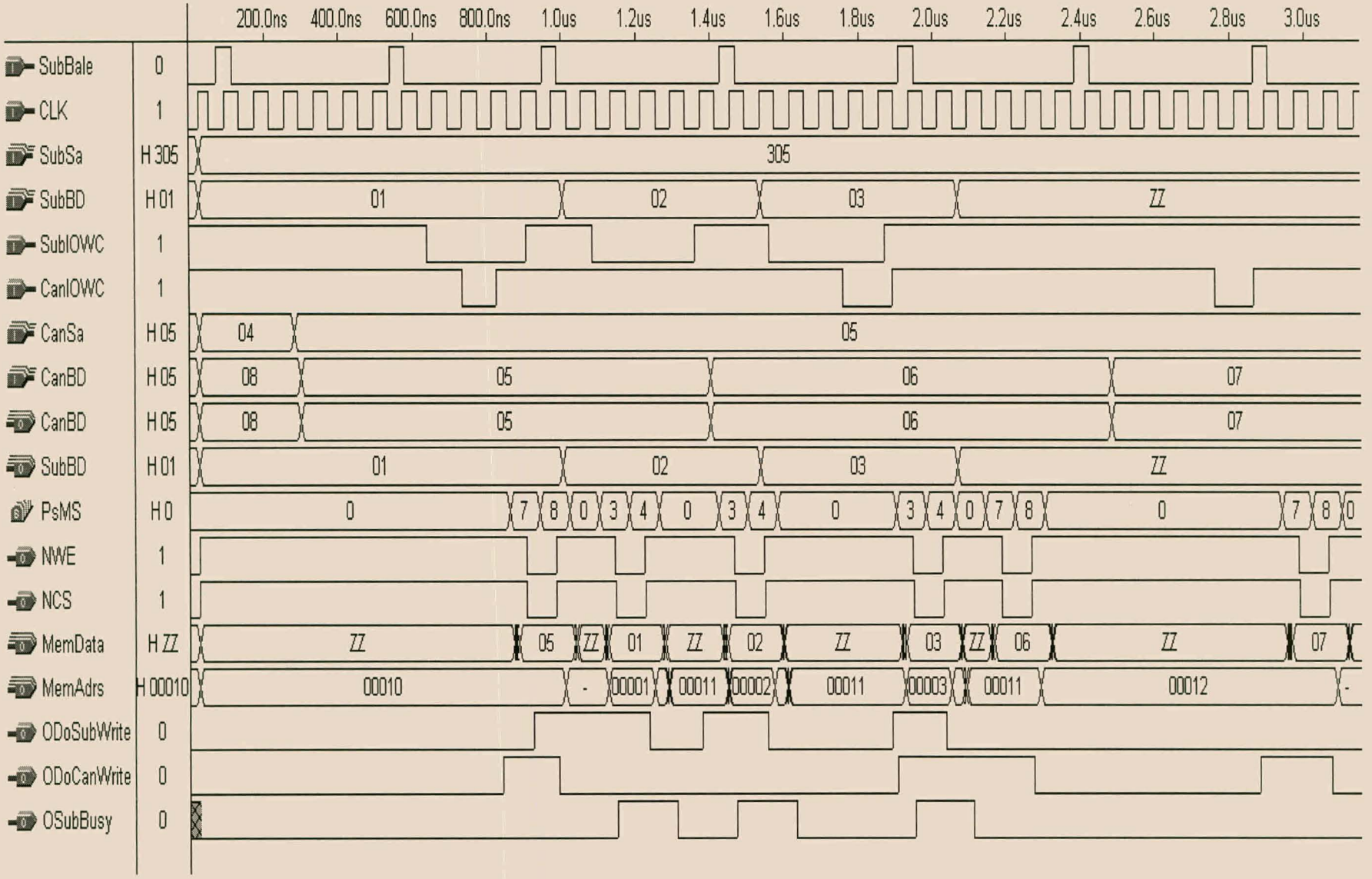
Vir skryfsiklusse vanaf die substelsel of die CAN stelsels bestaan daar twee registers waarin die data tydelik gestoor word, voordat dit in die geheue gestoor word. Sodra 'n data skryf vanaf die substelsel of die CAN stelsel plaasvind, word die korrekte vlaggie, DoSubWrite of DoCanWrite, gestel en die data word in die een van die twee skryfregisters gestoor. Die toestandsmasjien volg dan een van die twee skryf paaie en die waarde in die skryfregister word in die geheue gestoor. Aangesien die opdatering van die toestande op die afgaande rand van die klok geskied neem dit 3 kloksiklusse om 'n lees of skryfsiklus af te handel. Met 'n klokspoed van 12 MHz neem dit dus 250 ns om een geheue toegang af te handel.

Die instruksie wat die CAN mikroverwerker gebruik om die eksterne adres spasie aan te spreek staan bekend as 'n "MOVX" instruksie. So 'n instruksie duur twee masjiensiklusse. 'n Masjiensiklus beslaan 6 kloksiklusse en die totale tyd wat dit dus neem om 'n "MOVX" instruksie af te handel is dus (2 X 6)/(klok frekwensie). Met 'n klok frekwensie van 12 MHz is die tyd wat dit neem om 'n "MOVX" instruksie af te handel 1us. In die slegste geval, indien daar twee "MOVX" instruksies direk na mekaar voorkom, is die tempo wat die CAN verwerker met die geheue wil kommunikeer dus 1 "MOVX" instruksie elke 1 us.

Die lees en skryfsiklus van die ISA bus soos in figuur 9 aangegee, toon dat 'n lees of skryf instruksie 6 kloksiklusse duur. Met 'n klokfrekwensie van 12 MHz is die maksimum tempo waarteen die persoonlike rekenaar met die geheue kan kommunikeer dus 12MHz/6 = 2 MHz, of een geheue toegang elke 500 ns.

Indien die twee stelsels dus gelyktydig data na die geheue wil skryf hanteer die toestandsmasjien die arbitrasie. Dit word gedoen deur die toestandsmajien wat die geheue toegang versoek registreer sodra een van stelsels toegang tot die geheue verlang en die versoek so spoedig moontlik diens. Aangesien die spoed van die FPGA geheue toegang vinnig genoeg is, kan dit gelyktydige geheue toegang van beide stelsels hanteer.

Die MAXPLUS tydsimulasie vir die ontwikkelde stelsel word in figuur 13 getoon. Die verduideliking vir die seinname word in tabel 4 aangegee.



Figuur 13: Simulasie van gelyktydige geheue toegang

Sein Naam:	Beskrywing:
SubBale	ISA bus ALE
Clk	Stelsel Klok
SubSa	Substelsel Adres
SubBD	Substelsel Data
SubIOWC	Substelsel Skryf Lyn
CanIOWC	CAN verwerker Skryf Lyn
CanSa	CAN verwerker Adres
CanBD	CAN verwerker Data
PsMs	Toestandsmasjien toestand
NWE	Geheue skryf selekteer lyn
NCS	Geheue selekteer lyn
MemData	Geheue datapoort
MemAdrs	Geheue adres poort
OdoSubWrite	Substelsel skryf vlaggie
OdoCanWrite	CAN skryf vlaggie
OsubBusy	Substelsel geheue toegang vlaggie

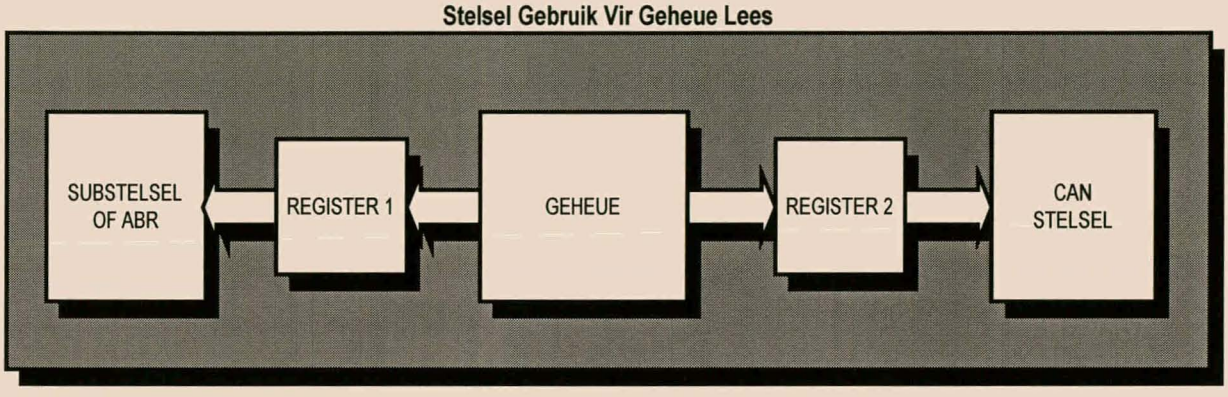
Tabel 4: Sein beskrywing

In die simulasie voer die CAN stelsel 3 skryfsiklusse uit. Die begin adres van die CAN stelsel geheue toegang is vir hierdie simulasie 10H en die waardes wat uitgeskryf word is 5, 6 en 7. Die CAN stelsel voer 'n skryfsiklus elke 1 us uit. Die substelsel skryf ook drie waardes na geheue met begin adres 1. Die substelsel voer 'n skryfsiklus elke 500 ns uit. Die sein PsMs dui die huidige toestand, soos in tabel 3 getoon, van die toestandsmasjien aan. Uit figuur 13 kan gesien word dat die stelsel wat ontwikkel is die gelyktydige skryfsiklusse van beide stelsels suksesvol kan hanteer.

Vir die huidige opstelling kan daar nie gewaarborg word dat indien albei stelsels gelyktydig 'n leessiklus uitvoer die data vanaf die geheue betyds, in terme van die opsteltyd voordat die lees lyne van die twee stelsels hoog gaan, vir albei stelsels beskikbaar sal wees nie. Vir die ISA geval bly die lees lyn vir vier en 'n half kloksiklusse laag, wat met 'n klokspoed van 12 MHz gelyk is aan 375 ns. Indien 'n geheue toegang dus deur die CAN stelsel begin word en direk daarna, binne 'n paar nano sekondes, word 'n leessiklus deur die ISA bus uitgevoer, is die toestandsmasjien nie in staat om aan die tydsvereistes van die ISA bus te voldoen nie. Die rede hiervoor is dat een

geheue toegang deur die toestandsmasjien 250ns duur en twee geheue siklusse nie binne 375ns afgehandel kan word nie.

Aangesien die toegang deur beide stelsels sekwensieel plaasvind deurdat boodskappe by agtereenvolgende adresse in die geheue gelees word, kon 'n stelsel ontwikkel word om die probleem te oorkom. Die stelsel wat gebruik word vir die leessiklus word in figuur 14 getoon.

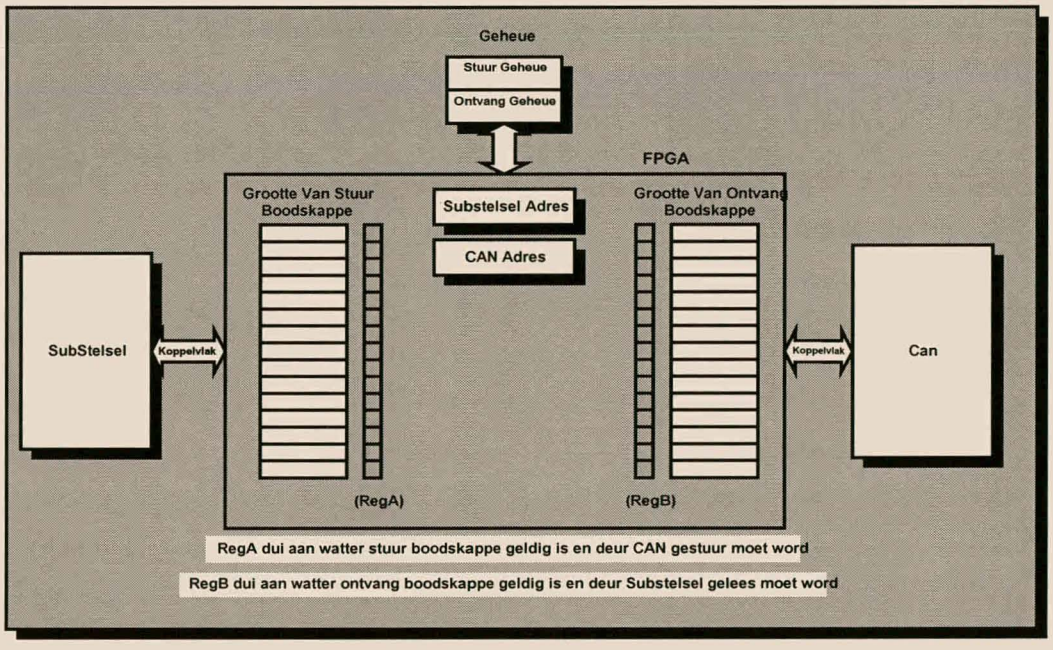


Figuur 14: Geheue Lees Stelsel

Twee registers bestaan waarin die data tydelik gestoor word. Sodra 'n stelsel 'n leessiklus uitvoer, word die data in die register aan die substelsel oorgedra. Die toestandsmasjien lees ook die volgende waarde uit die geheue en stoor dit in die register sodra die registerwaarde aan die stelsel toegeken is. Indien 'n stelsel dus 'n boodskap uit die geheue wil lees, moet dit eers 'n fop lees uitvoer, aangesien die waarde in die register 'n vorige geheue lees se waarde weerspieël. Met die opeenvolgende leessiklusse word die boodskap dan via die register uit die geheue gelees.

3.1.2.4 Geheue Spasie Bestuur:

Die fisiese blok geheue word opgebreek in 2 blokke naamlik 'n blok vir stuur boodskappe en 'n blok vir ontvang boodskappe. Hierdie 2 blokke word dan weer onderverdeel in kleiner blokke van 256 grepe vir die stoor van verskillende boodskappe. Die grootte van 256 grepe per blok kan aangepas word om aan die vereistes van die tipe data wat oor die netwerk gestuur word te voldoen. Vir elke blok bestaan daar ook twee registers in die FPGA wat die grootte en die destinasie of die bron adres aandui. Twee registers RegA en RegB dui ook aan watter van die blokke geldige boodskappe bevat. 'n Voorstelling van die stelsel word in figuur 15 getoon.



Figuur 15: Geheue Spasie Bestuur Stelsel

Sodra die substelsel 'n nuwe boodskap in die geheue wil skryf, lees dit die register RegA om te bepaal of daar 'n oop posisie in die stuur geheue is. Indien wel, selekteer die substelsel hierdie blok geheue en skryf die boodskap deur die FPGA na die geheue. Die FPGA is verantwoordelik vir die inkrementasie van die geheue posisie. Die grootte en destinasie adres word ook deur die substelsel na registers binne die FPGA geskryf. Sodra die volledige boodskap in die geheue ingeskryf is, genereer die FPGA 'n onderbreking by die CAN verwerker wat dan die nuwe boodskap weer uit die geheue kan lees.

Vir die ontvangs van boodskappe stoor die CAN verwerker die data van 'n boodskap in 'n oop blok ontvang geheue. Wanneer 'n boodskap volledig ontvang is, word die

ooreenstemmende bis in register RegB gestel, wat aandui dat daar 'n geldige boodskap in daardie blok geheue is. Die CAN verwerker stoor ook die oorsprong adres en die grootte van die boodskap in registers in die FPGA. 'n Onderbreking word dan by die substelsel genereer wat aandui dat 'n nuwe boodskap beskikbaar is om uit die ontvang geheue gelees te word. Wanneer die substelsel dan die boodskap uit die geheue gelees het, word die korrekte bis in register RegB gewis om die blok geheue weer beskikbaar te stel vir 'n nuwe ontvang boodskap.

3.1.2.5 Ander stelsels geïmplementeer op FPGA:

- Registers vir die stoor van die grootte van die boodskappe.
- Registers vir die stoor van adresse van die boodskappe.
- Registers vir die stoor van die berekende SOT waardes.
- Logika vir die opwek van onderbrekings by die substelsel sowel as die CAN verwerker.

3.1.3 Totale FPGA Stroomverbruik:

Die totale stelsel neem ongeveer 970 logiese selle van die FPGA in beslag. Volgens die formules van Altera sal die geskatte stroomverbruik vir die logika ongeveer 133 mA wees. Die berekening word in bylae 6 aangegee.

3.2 CAN VERWERKER:

3.2.1 Keuse van CAN verwerker:

Vir die implementasie van die CAN verwerker is daar gesoek na 'n standaard 8051 struktuur met 'n ingeboude CAN eenheid. Drie verwerkers is opgespoor en oorweeg vir die stelsel. Belangrike eienskappe van die drie verwerkers word in tabel 5 getoon.

Verwerker	Maks Klok Spoed	Instruksie Siklus tyd	Interne Program Geheue	Eksterne Program Geheue	Interne Data Geheue	Eksterne Data Geheue	Operasionele Stroom-verbruik	Luier Stroom-verbruik	Bystands Stroom-Verbruik
Philips P8xC592	16 MHz	750 us @ 16 MHz	16K grepe	64K grepe	2X256 grepe	64K grepe	50 mA @ 16 MHz	15 mA @ 16 MHz	150 uA @ 16 MHz
National Semi-Conductor COP884	10 MHz	1 us	2K grepe	-	64 grepe	-	15 mA @ 10 MHz	5.6 mA @ 10 MHz	450 uA @ 10 MHz
Infineon C505C	20 Mhz	300 ns @ 20 MHz	16K grepe	64K grepe	512 grepe	64K grepe	32 mA @ 20MHz	17.8 mA @ 20 MHz	50 uA @ 20 MHz

Tabel 5: Vergelyking van CAN verwerkers

Die COP 844 BC se CAN eenheid kan slegs 2 grepe per boodskap teen 1 Mbisse per sekonde stuur en aangesien die stelsel vir data kommunikasie gebruik wil word, is dit belangrik dat die maksimum aantal datagrepe, naamlik 8, teen die maksimum tempo van 1 Mbisse per sekonde gestuur word. Hierdie verwerker is dus nie verder vir implementasie oorweeg nie.

Die Infineon verwerker se CAN eenheid is meer gevorderd in terme van boodskap filtrering as die Philips verwerker. Die Infineon het ook 'n korter instruksie tyd 300 ns in vergelyking met die 750 us van die Philips verwerker. Die gemiddelde van die luier en bystands stroomverbruik van die Infineon verwerkers is ook laer as die van die Philips verwerker. Die Infineon verwerker is dus gekies vir die implementasie. 'n Oorsig van die CAN eenheid van die C505C word in bylae 7 aangegee.

3.2.2 Bespreking van CAN verwerker:

Die FPGA stel die CAN verwerker deur middel van 'n onderbreking in kennis dat daar 'n boodskap in die geheue is wat gestuur moet word. Die CAN verwerker moet dan eerstens vasstel watter boodskap in die geheue gestuur moet word. Daarna moet daar vasgestel word waarheen die boodskap gestuur moet word en of die ander stasie die boodskap op daardie stadium kan ontvang. Die grootte van die boodskap wat gestuur word moet ook vooraf aan die ander stasie oorgedra word. Indien die ander stasie wel bereid is om die boodskap te ontvang kan die boodskap raam vir raam, 8 datagrepe per raam, versend word.

Sodra 'n nodus 'n boodskap oor die netwerk ontvang dat daar 'n ander stelsel is wat 'n boodskap aan daardie stelsel wil oordra, moet die stelsel eers bepaal of daar plek in die geheue bestaan om 'n nuwe boodskap te stoor. 'n Boodskap word teruggestuur aan die oorspronklike stasie wat die boodskap wil versend wat aandui of die data versending wel kan plaasvind of nie. Wanneer die fisiese boodskap dan ontvang word moet die data na die ontvang van elke raam van die boodskap aan die FPGA oorhandig word, wat dit in die geheue stoor.

Om bostaande te implementeer word daar van 2 verskillende tipe rame gebruik gemaak naamlik:

1. Informasie rame bestaande uit datastuuraanvraag rame en antwoord op die datastuuraanvraag rame.
2. Data rame.

Die datastuuraanvraag raam word versend deur 'n nodus wat data het om aan 'n ander nodus oor te dra. Hierdie raam bevat die oorsprong en destinasie adres en die grootte van die data wat die nodus wil oordra.

In antwoord op die datastuuraanvraag raam word 'n raam teruggestuur wat weereens die oorsprong en destinasie adres bevat sowel as 'n antwoord of die data oordrag aanvraag. Indien 'n positiewe antwoord ontvang word kan daar voortgegaan word met die versending van die data met 8 datagrepe per pakkie.

3.2.2.1 Raamuitleg en Adressering:

Elke boodskap in die CAN protokol word uitgeken deur 'n 11 bis of 'n 29 bis identifiseerder. Hierdie identifiseerder dui ook die prioriteit van die boodskap aan. Hoe laer die waarde van die identifiseerder hoe hoër is die prioriteit van die boodskap. 2032 Verskillende boodskappe kan met die 11 bis identifiseerder versend word en met die 29 bis kan 2^{29} verskillende identifiseerders geskep word. Verskeie nodusse kan 'n boodskap met 'n sekere identifiseerder ontvang, maar slegs een nodus kan 'n raam met 'n sekere identifiseerder uitstuur. Dit verskil drasties van 'n stelsel waar elke nodus 'n unieke adres het. 'n Adresseringskema moet dus uitgewerk word sodat elke nodus met elke ander nodus kan kommunikeer, met inagneming van die prioriteite. Die vereistes vir die adresseringskema vir 'n algemene data kommunikasie netwerk is egter nou gekoppel aan die hoeveelheid nodusse op die netwerk. Aangesien die CAN protokol voorsiening maak vir 2 lengtes identifiseerders, naamlik 11 en 29, moet die bes moontlike opsie gekies word vir die spesifieke toepassing. Voordat die adresseringskema dus gefinaliseer word, moet die aantal nodusse op die netwerk dus rofweg bekend wees.

Ter illustrasie is 'n adresseringskema ontwikkel vir 'n maksimum van 15 nodusse op die netwerk. Die ontwikkelde adresseringskema vir die stelsel word in tabel 6 getoon.

		Prioriteit:		Adres Na:				Tipe:	Adres Van:			
Identifiseerder →		10	9	8	7	6	5	4	3	2	1	0
Nodus 1:	Begin:	P1	P2	0	0	0	0	0	0	0	0	0
	Eindig:	P1	P2	0	0	0	0	1	1	1	1	1
Nodus 2:	Begin:	P1	P2	0	0	0	1	0	0	0	0	0
	Eindig:	P1	P2	0	0	0	1	1	1	1	1	1
Nodus 3:	Begin:	P1	P2	0	0	1	0	0	0	0	0	0
	Eindig:	P1	P2	0	0	1	0	1	1	1	1	1
Nodus 4:	Begin:	P1	P2	0	0	1	1	0	0	0	0	0
	Eindig:	P1	P2	0	0	1	1	1	1	1	1	1
Nodus 5:	Begin:	P1	P2	0	1	0	0	0	0	0	0	0
	Eindig:	P1	P2	0	1	0	0	1	1	1	1	1
Nodus 6:	Begin:	P1	P2	0	1	0	1	0	0	0	0	0
	Eindig:	P1	P2	0	1	0	1	1	1	1	1	1
Nodus 7:	Begin:	P1	P2	0	1	1	0	0	0	0	0	0
	Eindig:	P1	P2	0	1	1	0	1	1	1	1	1
Nodus 8:	Begin:	P1	P2	0	1	1	1	0	0	0	0	0
	Eindig:	P1	P2	0	1	1	1	1	1	1	1	1
Nodus 9:	Begin:	P1	P2	1	0	0	0	0	0	0	0	0
	Eindig:	P1	P2	1	0	0	0	1	1	1	1	1
Nodus 10:	Begin:	P1	P2	1	0	0	1	0	0	0	0	0
	Eindig:	P1	P2	1	0	0	1	1	1	1	1	1
Nodus 11:	Begin:	P1	P2	1	0	1	0	0	0	0	0	0
	Eindig:	P1	P2	1	0	1	0	1	1	1	1	1
Nodus 12:	Begin:	P1	P2	1	0	1	1	0	0	0	0	0
	Eindig:	P1	P2	1	0	1	1	1	1	1	1	1
Nodus 13:	Begin:	P1	P2	1	1	0	0	0	0	0	0	0
	Eindig:	P1	P2	1	1	0	0	1	1	1	1	1
Nodus 14:	Begin:	P1	P2	1	1	0	1	0	0	0	0	0
	Eindig:	P1	P2	1	1	0	1	1	1	1	1	1
Nodus 15:	Begin:	P1	P2	1	1	1	0	0	0	0	0	0
	Eindig:	P1	P2	1	1	1	0	1	1	1	1	1

Tabel 6: 'n Moontlike adresseringskema vir maksimum 15 nodusse

Die stelsel maak van die standaard CAN, 11 bis identifiseerder gebruik. Met hierdie stelsel kan elk van die stasies boodskappe van 4 verskillende prioriteite uitsaai. Indien die prioriteit geïgnoreer word, sal sekere nodusse se boodskappe altyd hoër prioriteit geniet as ander nodusse met 'n hoër identifiseerder waarde. Dit is 'n ongewenste situasie en die prioriteit moet dus in die ontwikkeling van die stelsel in ag geneem word.

Die prioriteit van die boodskap word aangedui deur die bisse P1 en P2. Die hoogste prioriteit kom voor indien beide P1 en P2 nul is en die laagste prioriteit as beide een is. Die 2 adresvelde dui die oorsprong en destinasie van elke boodskap aan. Die tipe veld dui aan of 'n data of informasie raam versend word.

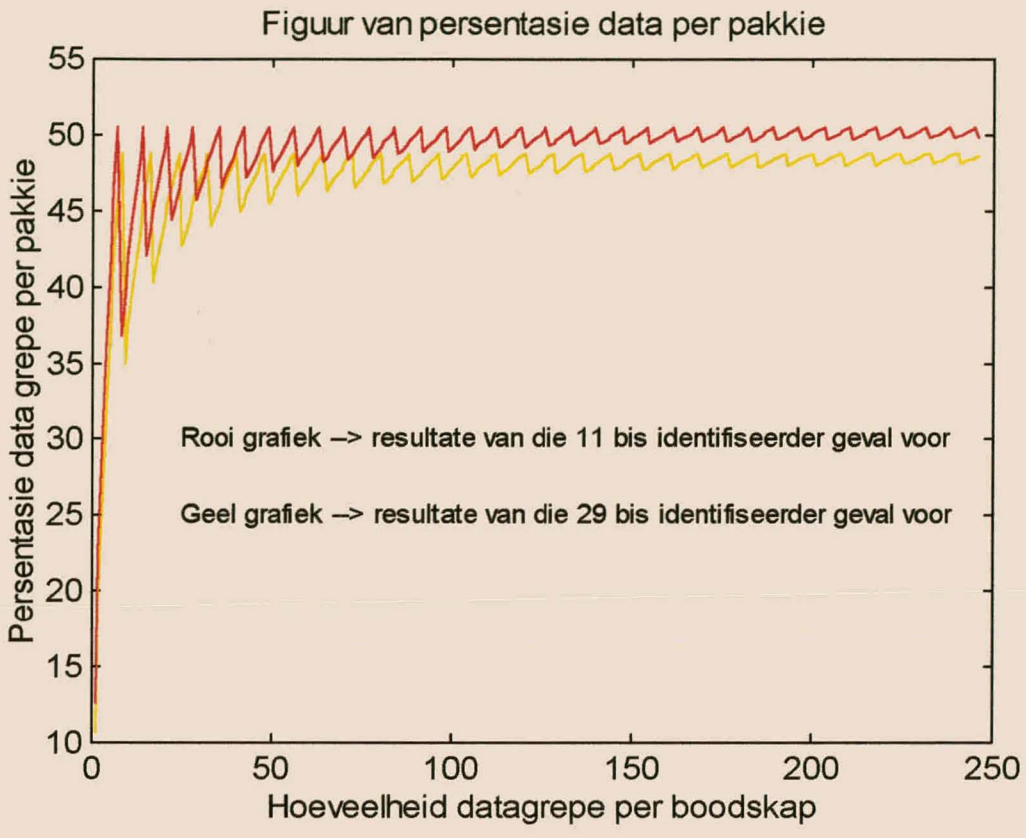
Bostaande stelsel kan nie gebruik word indien daar meer as 15 nodusse op die netwerk voorkom nie. Vir 'n netwerkstelsel waar daar tussen 15 en 31 nodusse op die netwerk voorkom kan die tipe en een van die prioriteits bisse uit die identifiseerder verwyder word. Dit stel 5 bisse beskikbaar vir die sender en die destinasie adres onderskeidelik en 1 bis vir die prioriteit. Die eerste data greep van die raam moet egter gebruik word om die raam tipe aan te dui. Met hierdie stelsel kan daar dus net 7 datagrepe per pakkie versend word en net 2 verskillende prioriteite kom voor in vergelyking met die 4 van die vorige stelsel.

Vir 'n stelsel met tussen 32 en 256 nodusse is daar 2 moontlike adresseringskemas.

Vir die eerste stelsel word die 11 bis identifiseerder opsie gebruik en slegs die destinasie adres, die tipe en die prioriteit word in die identifiseerder aangetoon. Die destinasie adres beslaan 8 bisse, die tipe, data of inligting, 1 bis en die oorblywende 2 bisse van die 11 bis identifiseerder word vir prioriteits doeleindes gebruik word. Die oorsprong adres moet egter in die data veld van die pakkie aangegee word. Dit stel slegs 7 grepe per pakkie beskikbaar vir data oordrag.

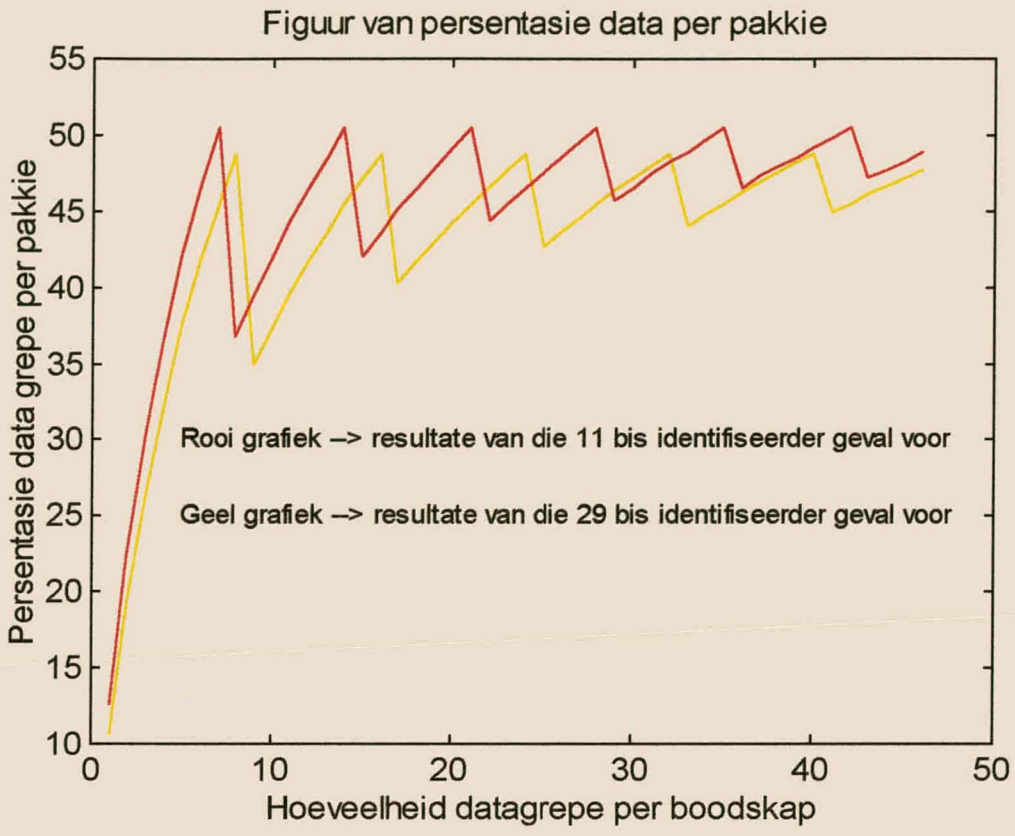
'n Tweede moontlike oplossing vir tussen 32 en 256 nodusse is om van die 29 bis identifiseerder gebruik te maak. Beide oorsprong en destinasie adres, die prioriteit en die tipe van die pakkie kan in die identifiseerder aangetoon word. Al 8 datagrepe kan dan vir data oordrag gebruik word.

Watter een van die bogenoemde 2 stelsels gekies word is afhanklik van die hoeveelheid data wat op 'n keer versend wil word. In die onderstaande figuur word die persentasie data per pakkie vir beide die 29 bis sowel as die 11 bis identifiseerder geval aangetoon vir verskillende boodskap groottes. Die totale pakkie grootte vir die 11 bis identifiseerder geval, met geen bis inplasing, is 111 bisse. Met 'n maksimum van 7 datagrepe per raam is die maksimum persentasie data per raam 50.45 %. Vir die 29 bis identifiseerder geval is die pakkie grootte 131 bisse en die maksimum hoeveelheid datagrepe per pakkie 8. Die maksimum persentasie datagrepe per raam is met bogenoemde syfers 48.9 %.



Figuur 16: Persentasie data per pakkie(1)

Indien die hoeveelheid data wat versend word wissel in grootte kan daar gesien word dat die stelsel wat van die 11 bis identifiseerder gebruik maak die beste funksioneer. Figuur 16 toon die effektiwiteit van die 2 stelsels met die hoeveelheid data wat versend word wat wissel tussen 0 en 246 datagrepe. Die geel grafiek stel die resultate van die 29 bis identifiseerder voor en die rooi grafiek die resultate van die 11 bis identifiseerder. Figuur 17 toon die persentasie data per pakkie vir 'n datagrepe tussen 0 en 46. Uit die figuur kan gesien word dat indien die hoeveelheid datagrepe wat versend word altyd 8, 16, 24 of 32 is, is die effektiwiteit van die 29 bis stelsel beter. Vir algemene data kommunikasie is die hoeveelheid data wat per boodskap versend word egter nie vasgestel nie en vir 'n veranderlike hoeveelheid data funksioneer die 11 bis identifiseerder stelsel beter.



Figuur 17: Persentasie data per pakkie(2)

3.2.2.3 Intydse Aspek:

Om te verseker dat boodskappe nie verlore gaan nie, moet daar aan sekere tydvereistes voldoen word. Die tyd bestee aan die verwerking van enige boodskap moet korter wees as die tyd wat dit neem om 2 boodskappe direk na mekaar te ontvang. Vervolgens word die maksimum tyd beskikbaar vir die verwerking van boodskappe bepaal.

Die bus spoed is 1 Mbisse/Sek. Die lengte van die kortste moontlike raam wat versend word hang af van die adresseringskema wat gebruik word. Die kleinste pakkie grootte kom voor indien die tipe en beide oorsprong en destinasie adresse in die 11 bis identifiseerder aangetoon word en indien slegs een data greep gestuur word. Die minimum pakkie grootte vir so 'n pakkie is 55 bisse. Die tyd beskikbaar vir die verwerking van data wat ontvang is oor die netwerk, is dus die tyd wat dit neem om 55 bisse teen 1M bisse per sekonde te stuur oor die netwerk en dit is gelyk aan 55 us. Met 'n klok frekwensie van 20 MHz neem die C505C 300 ns om 1 masjiensiklus af te handel en volgens die datavel voer 58% van die instruksies in een masjiensiklus uit. Daar kan dus min of meer 183 (55us/300ns) instruksies uitgevoer word voordat 'n volgende ontvang onderbreking genereer word deur 'n pakkie van die minimum lengte van 55

bisse. Boodskappe ontvang oor die netwerk moet dus binne 55 us verwerk word om te verhoed dat nuwe data oor 'n boodskap gekryf word wat nog nie verwerk is nie.

Die stelsel moet reageer op 4 verskillende onderbrekings naamlik 'n eksterne, stuur, ontvang of fout onderbreking. Die aksies wat uitgevoer moet word wanneer elkeen van die bogenoemde voorkom word vervolgens aangegee.

1. Eksterne Onderbreking:

Sodra 'n eksterne onderbreking voorkom is daar 'n nuwe boodskap om te stuur en moet die CAN verwerker 'n register binne die FPGA lees om te bepaal watter nuwe boodskap gestuur moet word. Die CAN verwerker lees dan die destinasie vir die boodskap vanaf die FPGA asook die grootte van die boodskap. 'n Datastuuraanvraag raam word dan uitgestuur na die destinasie nodus met die bron adres en die grootte van die boodskap ingesluit.

2. Ontvang Onderbreking:

Indien 'n ontvang onderbreking voorgekom het is een van 3 moontlike tipe rame ontvang:

2.1 Datastuuraanvraag Raam:

Indien hierdie raam ontvang is, moet die CAN stelsel bepaal of daar 'n oop posisie in die geheue is waar boodskap gestoor kan word. Indien daar wel plek in die geheue is moet daardie posisie reserveer word vir die boodskap. 'n Positiewe antwoord boodskap word dan ook aan die ander stasie gestuur wat aandui dat die versending van data kan voortgaan. Die adres en die grootte van die boodskap wat verwag word, word ook gestoor. Indien geen plek in die geheue beskikbaar is nie word 'n negatiewe antwoord boodskap aan die stasie gestuur.

2.2 Antwoord op Stuuraanvraag Raam:

Indien 'n positiewe boodskap ontvang is, kan daar voortgegaan word met die versending van die data. Afhangende van die adresseringskema wat gebruik word kan daar 7 tot 8 datagrepe per raam versend word. Na afloop van die versending van elke pakkie van die totale boodskap moet daar eers vasgestel word of daar nie ander tyds kritiese take ontstaan het wat afgehandel moet word voordat daar met die versending van die volgende pakkie van die boodskap voortgegaan kan word nie.

Indien 'n negatiewe boodskap ontvang is kan die stelsel na 'n sekere tydsverloop weer 'n Datastuuraanvraag raam stuur. Indien daar na 'n sekere hoeveelheid probeerslae geen positiewe boodskap ontvang word nie, kan daar 'n foutboodskap aan die substelsel oorgedra word.

2.3 Data Raam:

Sodra 'n data raam ontvang word moet die CAN verwerker dit aan die FPGA oordra sodat dit in die geheue gestoor kan word.

3. Stuur Onderbreking:

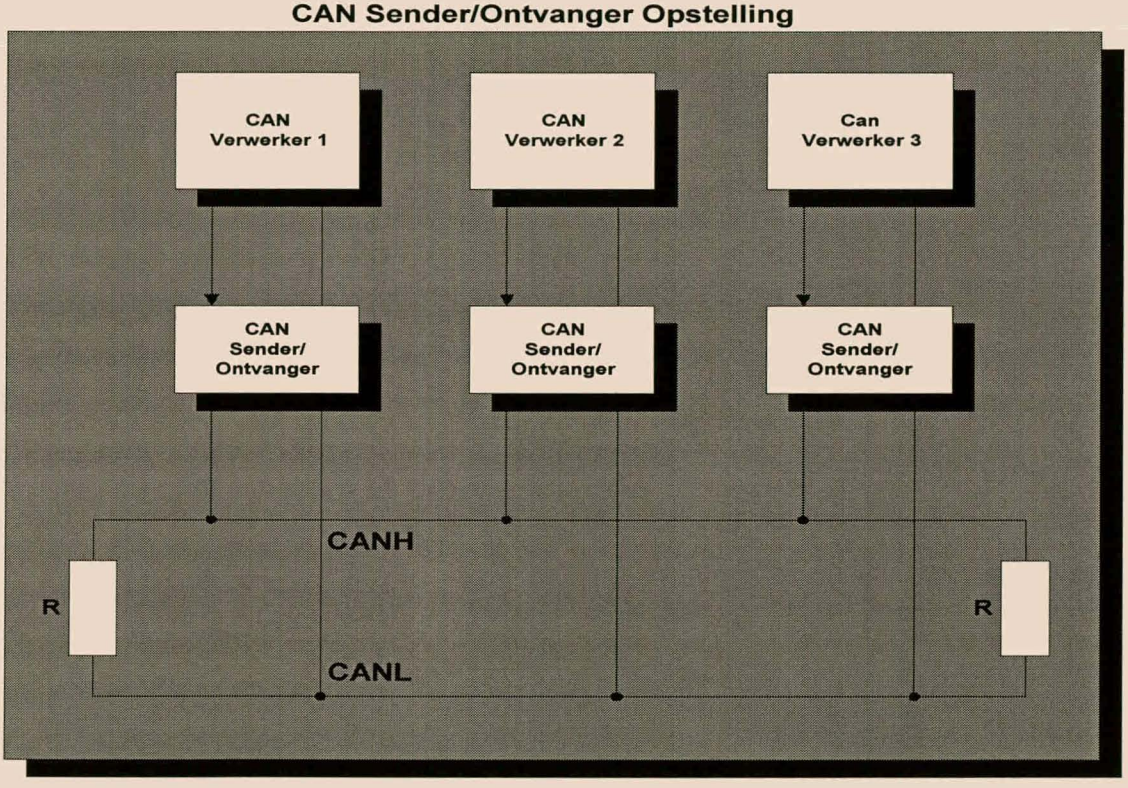
Indien 'n stuur onderbreking voorkom is die transmissie van 'n boodskap suksesvol afgehandel en kan daar voortgegaan word met die versending van die volgende boodskap.

4. Fout Onderbreking:

Sodra die CAN verwerker die "Bus Af" toestand binnegaan as gevolg van 'n reeks foute wat voortgekom het, moet die CAN eenheid van die verwerker van voor af geïnisialiseer word. Indien die "Bus Af" toestand elke keer na her inisilisering van die stelsel voorkom, is daar fout op die CAN stelsel en moet die fout gerapporteer word aan 'n oorhoofse stelsel.

3.3 CAN Sender/ontvangers:

Die doel van die sender/ontvanger is om as koppelvlak te dien tussen die protokol eenheid, in die geval die CAN verwerker, en die fisiese CAN bus. Die fisiese CAN bus het twee logiese toestande naamlik dominant en resessief. Die bus verkeer in die resessiewe toestand wanneer die dryf eenhede van al die sender/ontvangers wat aan die netwerk gekoppel is passief is. In hierdie toestand is die differensiële spanning tussen die twee bus lyne ongeveer nul. Indien enige van die drywers van die sender/ontvangers in die aktiewe toestand is, is die bus in die dominante toestand. Die dominante toestand word voorgestel deur 'n differensiële spanning van meer as 'n minimum drempel vlak.



Figuur 18: CAN Sender/Ontvanger opstelling

Die opstelling vir die sender/ontvanger stelsel word in figuur 18 getoon. Die sender/ontvangers het aan die eenkant 'n koppelvlak met die CAN verwerker. Die koppelvlak bestaan uit onder andere 2 data lyne, RxD en TxD, wat vir TTL vlak data transmissie gebruik word. Die sender/ontvanger skakel hierdie TTL vlak data ontvang vanaf die CAN verwerker om na 'n differensiële spanning wat dan op die CAN bus lyne CANH en CANL geplaas word. Netso word die differensiële spanning wat oor die netwerk lyne CANH en CANL ontvang word weer in TTL vlakke omgesit en aan die CAN verwerker gestuur deur middel van die RxD lyn.

Vir die fisiese implementasie is daar gesoek na komponente wat die maksimum CAN bus spoed van 1 Mbisse per sekonde kan hanteer. In die onderstaande tabel word die belangrikste eienskappe van die komponente wat gevind is aangetoon. Die belangrikste eienskappe van die sender/ontvangers is hul stroomverbruik en of hul in 'n bystands modus kan funksioneer.

Sender/ Ontvanger:	Verpakking:	Temperatuur min/maks:	Stroomverbruik Rescessiewe Toestand:	Stroomverbruik Dominante Toestand:	Bystands Modus:	Bystands Stroom Verbruik:
Infineon TLE 6250	8 pen "Surface Mount"	-40 tot 150 °C	15 mA	80 mA	Ja	200 uA
Unitrode UC5350	8 pen dip / 8 pen "Surface Mount"	-40 tot 125 °C	13 mA	70 mA	Ja	1 mA
Temic Si9200EY	8 pen "Surface Mount"	-40 tot 125 °C	25 mA	75 mA	Nee	--
Alcatel MTC-3054	16 pen "Surface Mount"	-55 tot 150 °C	14 mA	110 mA	Ja	300 uA
Philips PCA82C250	8 pen dip / 8 pen "Surface Mount"	-40 tot 125 °C	14 mA	70 mA	Ja	100 uA

Tabel 7: Vergelyking van CAN sender/ontvangers

Die 2 met die mees gewenste eienskappe is die Philips en die Unitrode komponente. Prysgewys is daar ook nie 'n groot verskil tussen die twee komponente nie, albei is in Suid-Afrika beskikbaar teen ongeveer R12.00 per eenheid. Waar die Philips komponent 'n laer bystands stroomverbruik het, het die Unitrode komponent 'n ietwat laer stroomverbruik in die resessiewe toestand. Die verskil tussen die twee is egter persentasie gewys heelwat kleiner in die geval van die resessiewe toestand stroomverbruik in vergelyking met die bystands stroomverbruik, waar die Philips komponent die beste werkverrigting toon. Vir die implementasie is die Philips komponent dus gekies.

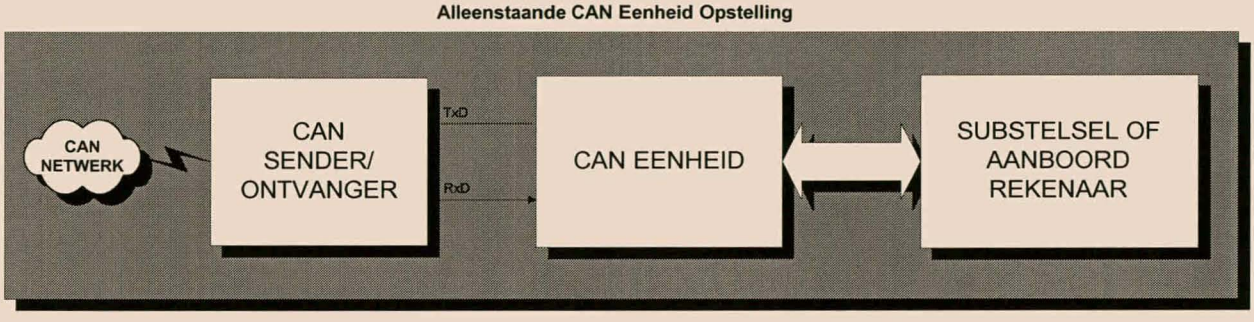
Hoofstuk 4: Stelsel 2: Alleenstaande CAN Eenheid

4.1 Agtergrond:

Een moontlike implementasie van die CAN protokol is deur middel van 'n alleenstaande CAN eenheid. Die feit dat hierdie alleenstaande CAN eenhede gebruik maak van 'n standaard koppelvlak, maak hierdie komponente ideaal vir gebruik, in terme van 'n generiese koppelvlak, vir 'n algemene kommunikasiestelsel. Die gebruik van so 'n alleenstaande CAN eenheid is ondersoek en word verder in hierdie hoofstuk bespreek.

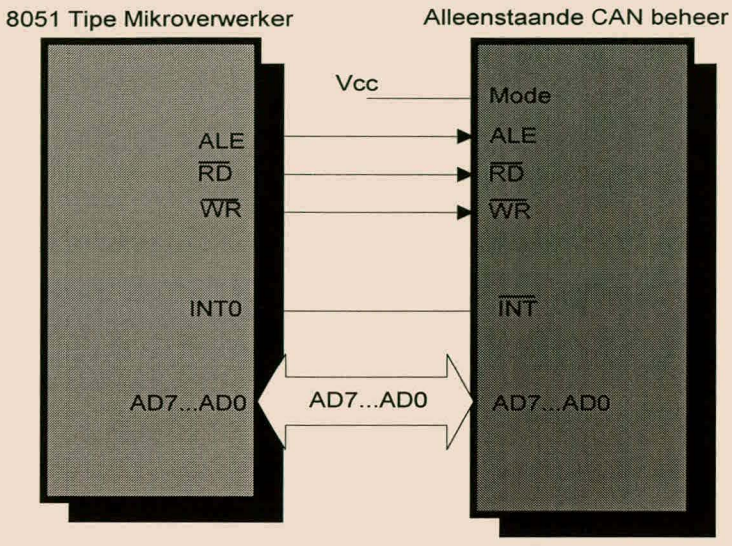
4.2 Werking:

Figuur 19 toon die opstelling vir 'n alleenstaande CAN verwerker.



Figuur 19: Alleenstaande CAN verwerker opstelling

Die verwerker wat die substelsel toepassing implementeer koppel deur middel van 'n standaard koppelvlak aan die CAN eenheid. Die CAN eenheid is verantwoordelik vir die implementasie van die CAN protokol en kommunikasie met die CAN sender/ontvanger.



Figuur 20: Koppeling tussen Alleenstaande CAN Beheerder en 'n 8051

Die tipiese opstelling vir die koppeling van 'n alleenstaande CAN beheerder aan 'n 8051 tipe mikroverwerker word in figuur 20 getoon. Die alleenstaande CAN beheerder word in die eksterne geheue ruimte van die 8051 gekarteer. Die 8051 kommunikeer met die CAN eenheid deur register by verskillende adresse aan te spreek. Die CAN eenheid stel die verwerker in kennis van veranderinge, soos byvoorbeeld 'n nuwe boodskap wat ontvang is, deur middel van die eksterne onderbrekings lyn van die verwerker.

4.3 Keuse van Alleenstaande CAN eenheid:

Verskillende alleenstaande CAN eenhede is ondersoek vir 'n moontlike implementasie in 'n algemene netwerkstelsel. Tabel 8 toon 'n opsomming van die verskillende eenhede.

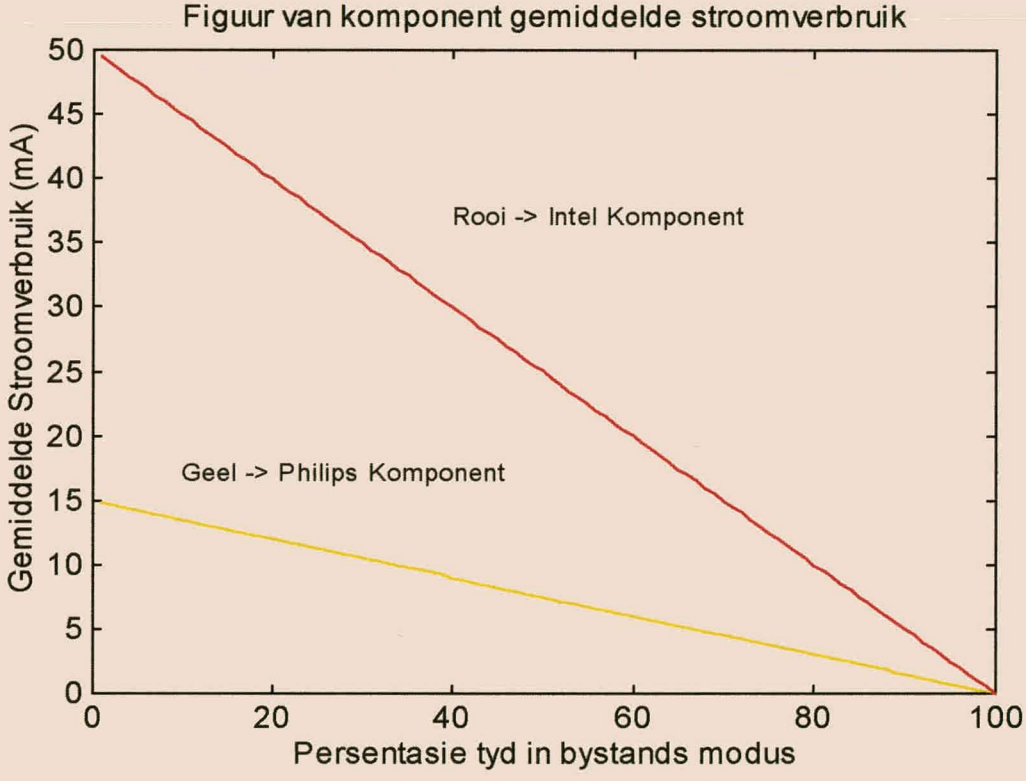
Komponent:	Tipe koppelvlak:	ntvang Buffers:	Stuur Buffers:	Maks Bus Spoed:	Verpakking:	Aktiewe Stroom Verbruik:	Bystands Stroom Verbruik:
Philips SJA1000	8 bis multipleks 8 bis nie-multipleks	8	1	1Mbisse/ Sek	28 pen	15 mA @24 MHZ	40uA
Infineon SAE 81C90	8 bis multipleks Serieel	0-15	0-14	1Mbisse/ Sek	44 of 28 pen	30 mA	Geen Bystands Modus
Intel 82527	8 bis multipleks 16 bis multipleks 8 bis nie-multipleks Serieel	0-15	0-14	1Mbisse/ Sek	44 pen	50mA @16MHz	25uA
Microchip MCP2510	Serieel	2	3	1Mbisse/ Sek	18pen	10mA @5MHz	20uA

Tabel 8: Vergelyking van Alleenstaande CAN beheerders

Aangesien die stelsel wat gekies word aan 'n wye verskeidenheid verwerkers moet koppel is die aantal verskillende koppelvlakke wat 'n komponent ondersteun belangrik. Die Microchip komponent is eerste geëlimineer as 'n moontlike kandidaat vir implementasie, aangesien dit slegs 'n seriële koppelvlak ondersteun. So 'n seriële koppelvlak is slegs in sommige gevalle die ideale oplossing en die gebrek aan 'n parallelle koppelvlak het die komponent nie geskik gemaak vir die gebruik in 'n generiese stelsel nie. Die Intel 82527 ondersteun die grootste verskeidenheid verskillende koppelvlakke, met 'n 16, 8, sowel as 'n seriële koppelvlak.

Wat die aktiewe stroomverbruik van die oorblywende 3 komponente betref vertoon die Philips komponent die beste. In die bystands modus is die stroomverbruik van die Intel komponent egter die laagste.

Die vraag is egter watter een van die Philips of die Intel komponent die laagste gemiddelde stroomverbruik toon. Dit is afhanklik van die persentasie tyd wat die komponent in die aktiewe of bystands modus deurbring. Aangesien die komponent aan 'n verskeidenheid verwerkers gekoppel sal word, wat verskillende hoeveelhede data stuur en ontvang, kan die persentasie tyd bestee in 'n sekere modus nie vooraf bepaal word nie. Die gemiddelde stroomverbruik van die 2 komponente word in figuur 21 geplot teenoor die persentasie tyd bestee in die bystands modus.



Figuur 21: Vergelyking van Intel en Philips komponent stroomverbruik

Aangesien die stroomverbruik soveel hoër is in die aktiewe as in die bystands modus lewer die bystands stroom 'n baie klein bydra tot die gemiddelde stroomverbruik. Die Intel komponent lewer slegs 'n laer gemiddelde stroomverbruik indien die komponent meer as 99.96% van die tyd in die bystands modus verkeer. In die algemeen is die gemiddelde stroomverbruik van die Philips komponent dus laer.

Om 'n keuse tussen die 2 komponente te maak, moet die voordele wat elkeen inhou vergelyk word. Indien die koppelvlak van die Philips komponent voldoende is vir 'n generiese stelsel sal hierdie komponent voorkeur geniet weens die gewenste

stroomverbruik. Die 2 komponente se koppelvlak eenhede verskil in die opsig dat die Intel komponent 'n seriële sowel as 'n 16 bis koppelvlak bevat wat die Philips komponent nie besit nie. Meeste verwerkers ondersteun egter 'n 8 bis koppelvlak soos teenwoordig in die Philips komponent. Die Philips SJA1000 komponent is dus gekies weens die laer stroomverbruik. 'n Beskrywing van die Philips SJA1000 word in bylae 8 gegee.

Verantwoordelikhede van verwerker gekoppel aan CAN eenheid:

1. Opstel van SJA1000 na herstel:

Na krag aan moet die verwerker die CAN eenheid in die regte modus plaas, die boodskap filter opstel en die korrekte waarde na die sinkronisasie registers skryf.

2. Opstel van stuur boodskappe:

Voordat 'n boodskap gestuur word moet die verwerker die boodskap, lengte van die boodskap, sowel as die identifiseerder na die CAN eenheid skryf. Die transmissie van die boodskap word begin sodra die verwerker die Transmissie Aanvraag bis in die CAN eenheid stel.

3. Lees van ontvangde boodskappe:

Indien die ontvang onderbreking wel geënisialiseer is, wek dit 'n onderbreking by die verwerker op sodra 'n boodskap suksesvol oor die netwerk ontvang is. Die Ontvang Buffer Status vlaggie word ook na 1 (vol) gestel. Die verwerker lees dan die boodskap uit die Ontvang Buffer en stuur 'n vrystel bevel aan die CAN eenheid. Hierdie bevel stel weer die Ontvang Buffer vry vir die stoor van nuwe boodskappe vanuit die EIEU (Eerste In Eerste Uit) buffer.

4. Foute wat verwerker moet hanteer:

Foute wat kan voorkom is die volgende:

4.1 Data Oorvloei:

Hierdie toestand kom voor indien die 64 greep EIEU buffer vol is en 'n nuwe boodskap word oor die netwerk ontvang wat ook in die EIEU buffer gestoor moet word. Hierdie toestand ontstaan indien die verwerker wat aan die CAN eenheid gekoppel is nie

genoeg tyd het om die boodskappe uit die EIEU buffer te lees nie. Om te verhoed dat geen boodskappe verlore gaan nie moet die stelsel so ontwerp word dat die verwerker altyd in staat is om die boodskappe uit te lees voordat die buffer vol is.

4.2 Bus Af toestand:

Indien die stuur of ontvang fout tellers 255 bereik gaan die CAN eenheid 'n Bus Af toestand binne en word geen boodskappe meer oor die netwerk gestuur of ontvang nie. Die verwerker word deur die CAN eenheid deur middel van 'n onderbreking in kennis gestel . Die verwerker moet dan die CAN eenheid herstel en van voor af opstel.

4.4 CAN Sender/ontvanger:

Die vereistes vir die CAN sender/ontvanger vir hierdie stelsel is dieselfde as in die geval van stelsel 1 en die keuse van die CAN sender/ontvanger word dus nie hier herhaal nie. Die sender/ontvanger wat die beste sal funksioneer in die gegewe stelsel is die Philips PCA82C250.

4.5 Tipe Rame:

Aangesien die verwerkers van die verskillende substelsels op die satelliet nie met 'n selfstandige eenheid, soos in die geval met stelsel 1, kommunikeer nie, is die substelsels self verantwoordlik vir die oorhoofse kommunikasie protokol wat gebruik word. Aangesien daar 'n verskeidenheid stelsels op die netwerk kan voorkom wat verskillende hoeveelhede data oor die netwerk wil versend, verskil die eienskappe van die rame wat oor die netwerk versend word ook drasties. Verskillende interne protokolle kan binne die CAN rame implementeer word, asook verskillende maniere vir die opstel en afbreek van kommunikasie kanale. Twee stelsels wat met mekaar kommunikeer moet net aan dieselfde stel reëls voldoen.

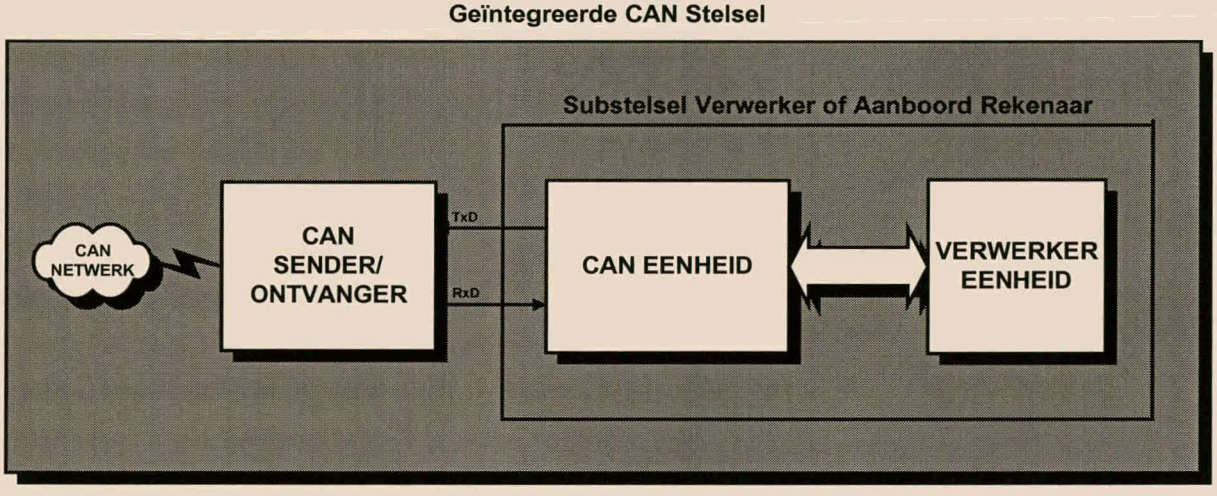
4.6 Adresseringskema:

Die adresseringskema wat gebruik word moet regoor die stelsel dieselfde formaat hê sodat adressering suksesvol kan plaasvind. Soos in die geval met stelsel 1 is die adresseringskema wat gekies word afhanklik van die hoeveelheid nodusse op die netwerk. Dieselfde adresseringskemas wat oorweeg is vir stelsel 1 kan ook vir hierdie stelsel oorweeg word, behalwe dat die raamtipe verwyder kan word.

Hoofstuk 5: Stelsel 3: Aanboord CAN Eenheid

5.1 Agtergrond:

Geïntegreerde CAN beheerders word ook algemeen gebruik vir die implementasie van die CAN protokol. 'n Geïntegreerde CAN stelsel bestaan uit 'n CAN eenheid wat koppel aan die verwerker eenheid. 'n Groot voordeel van hierdie implementasie is dat slegs 'n eksterne CAN sender/ontvanger nodig is. Indien so 'n stelsel gebruik kan word om die substelsel of ABR funksie te verrig, sowel as die netwerk protokol hantering sal dit 'n ideale oplossing wees in terme van 'n lae komponent telling en ook 'n lae stroomverbruik. Figuur 22 toon die opstelling vir 'n geïntegreerde CAN verwerker.



Figuur 22: Geïntegreerde CAN Stelsel

5.2 Keuse van Verwerker:

Verskillende verwerkers kan nie vooraf oorweeg word vir implementasie nie, aangesien die verskillende stelsels op die satelliet verskillende tipe verwerkers verlang. Die CAN eenheid van 'n nuwe verwerker wat egter tot die stelsel toegevoeg word, moet egter eers evalueer word vir aanpasbaarheid tot die stelsel. Nie alle verwerker CAN eenhede kan byvoorbeeld die maksimum bus spoed van 1 Mbisse per sekonde hanteer of 8 datagrepe per pakkie versend nie.

5.3 Verantwoordelikheid van verwerker:

Die verantwoordelikhede van die verwerker in terme van CAN kommunikasie is dieselfde as in die geval van stelsel 2. Al verskil is dat die CAN eenheid hier binne die verwerker voorkom waar in die geval van stelsel 2 die CAN eenheid 'n aparte

komponent is. Die toegang tot die CAN eenheid is egter in sommige gevalle vinniger vir die interne CAN eenheid in vergelyking met die eksterne CAN eenheid.

5.4 CAN Sender/ontvanger:

Die vereistes vir die CAN sender/ontvanger vir hierdie stelsel is dieselfde as in die geval van stelsel 1 en 2 en die keuse van die CAN sender/ontvanger word dus nie hier herhaal nie. Die sender/ontvanger wat die beste sal funksioneer vir die spesifieke stelsel is die Philips PCA82C250.

5.5 Tipe Rame en Adresseringskema:

Net soos die geval met stelsel 2 kan verskillende interne protokolle binne die CAN rame implementeer word. Verskillende maniere vir die opstel en afbreek van kommunikasie kanale kan ook bestaan. Enige twee stelsels wat met mekaar kommunikeer moet net aan dieselfde stel reëls voldoen en alle stelsels moet dieselfde adresseringskema gebruik Dieselfde adresseringskemas wat oorweeg is vir stelsel 1 en stelsel 2 kan ook vir hierdie stelsel oorweeg word.

Afdeling 2: Evaluering van stelsels

- ✓ Hoofstuk 6: Vergelyking van stelsels
- ✓ Hoofstuk 7: Gevolgtrekking

Hoofstuk 6: Vergelyking van Stelsels

Elkeen van bogenoemde stelsels het sekere voordele bo die ander stelsels en om 'n geskikte stelsel te kies moet die stelsels se eienskappe met mekaar vergelyk word. Sekere eienskappe van die stelsels wat as belangrik geag is, word afsonderlik vergelyk.

6.1 Komponent telling:

Die komponent telling word geneem as die hoeveelheid ekstra komponente wat by die stelsel verwerker gevoeg moet word om netwerktoegang moontlik te maak.

Stelsel:	1:	2:	3:
Komponent Telling:	4	2	1

Tabel 9: Vergelyking van komponent tellings

Kommentaar:

Stelsel 1 bestaan uit 'n FPGA, geheue, CAN verwerker en 'n CAN sender/ontvanger. Van al die stelsels is die komponent telling vir hierdie stelsel die hoogste. Stelsel 2 bestaan uit die CAN eenheid en die CAN sender/ontvanger terwyl stelsel 3 slegs 'n eksterne sender/ontvanger verlang. Intern moet die verwerker van stelsel 3 egter 'n CAN eenheid hê.

6.2 Koste:

Stelsel 1:

Komponent:	Prys:
FPGA	R180
Geheue	R250
CAN Sender/ontvanger	R12
CAN Verwerker	R50
Totaal	R462

Tabel 10: Stelsel 1 koste

Stelsel 2:

Komponent:	Prys:
CAN Sender/ontvanger	R12
Alleenstaande CAN Eenheid	R40
Totaal	R52

Tabel 11: Stelsel 2 koste

Stelsel 3:

Dit is moeilik om die prys te bepaal vir hierdie stelsel aangesien die CAN eenheid en die stelsel verwerker 'n eenheid vorm.

Komponent:	Prys:
CAN Sender/ontvanger	R12
Toename in Verwerker prys a.g.v. Interne CAN Eenheid	?
Totaal	?

Tabel 12: Stelsel 3 koste

Vir byvoorbeeld die C505 is daar geen prys verskil tussen die standaard verwerker en die verwerker met die aanboord CAN eenheid nie.

Kommentaar:

Dit is moeilik om die goedkoopste stelsel te bepaal weens die feit dat die prys van stelsel 3 afhang van die substelsel verwerker wat gekies word. Wat wel gesien kan word is dat stelsel 2 baie goedkoper is as stelsel 1 en as die C505C as voorbeeld geneem word, sal stelsel 3 ook 'n koste effektiewe oplossing bied.

6.3 Betroubaarheid:

Om enige van die stelsels se betroubaarheid te verhoog kan die stelsels gedupliseer word in 'n meester/slaaf opstelling. Sodra daar bemerk word dat enige stelsel nie meer korrek funksioneer nie, kan daar oorgeskakel word na die sekondêre eenheid. Die primêre eenheid word dan afgeskakel en die sekondêre eenheid neem die funksionaliteit van die primêre eenheid oor.

Hoe meer komponente daar in 'n stelsel voorkom, sonder enige duplisering, hoe groter is die kans dat daar fout kan gaan met een van die komponente. Die koste verbonde aan die duplisering van 'n stelsel vir 'n meester/slaaf opstelling vir 'n stelsel met meer komponente is ook groter.

In hierdie opsig vertoon stelsel 1 swak in vergelyking met die ander 2 stelsels in terme van koste van duplisering en komponent telling. Vir stelsels 2 en 3 kan die duplisering vir hoër betroubaarheid redelik maklik gedoen word.

6.4 Stroomverbruik:

Al die stelsels het 'n CAN sender/ontvanger en vir die vergelyking van die stelsels word die CAN sender/ontvanger geïgnoreer.

Stelsel 1:

Komponent:	StroomVerbruik:
FPGA	133 mA
Geheue	60 mA
CAN Verwerker	32 mA
Totaal	225 mA

Tabel 13: Stelsel 1 stroomverbruik

Stelsel 2:

Komponent:	StroomVerbruik:
Alleenstaande CAN Eenheid	15 mA

Tabel 14: Stelsel 2 stroomverbruik

Stelsel 3:

Indien die CAN sender/ontvanger geïgnoreer word, het stelsel 3 geen ekstra komponente nie. Daar is egter 'n toename in stroom, want 'n verwerker met 'n interne CAN eenheid moet gebruik word. Die toename in stroom as gevolg van die interne CAN eenheid is moelik om te bepaal. Die funksionaliteit van die interne CAN eenheid is egter ongeveer dieselfde as 'n alleenstaande CAN eenheid en die toename in stroomverbruik behoort nie meer te wees as die stroomverbruik van 'n alleenstaande CAN eenheid nie. Die toename in stroomverbruik word dus vir evaluasie doeleindes geneem as die van 'n alleenstaande CAN eenheid (Philips SJ1000).

Komponent:	StroomVerbruik:
Interne CAN Eenheid	15 mA

Tabel 15: Stelsel 3 stroomverbruik

Kommentaar:

Die stroomverbruik van stelsels 2 en 3 is 15 keer laer as die stroomverbruik van stelsel 1.

6.5 Stelsel verwerkerlas:

Die verwerkerlas hang af van hoe die opstelling, fout hantering en boodskap stuur en ontvang deur die stelsel hanteer word. Die hoeveelheid keer wat die substelsel verwerker onderbreek word om 'n boodskap van 'n sekere lengte tussen die netwerkstelsel en die verwerker oor te dra beïnvloed ook die verwerkerlas.

Stelsel 1:

Die substelsel verwerkerlas is die laagste in die geval van stelsel 1. Die substelsel verwerker word slegs onderbreek indien volledige boodskappe in die geheue is wat deur die verwerker uitgelees moet word. Die fouthantering en alle ander transmissieprotokol is nie vir die substelsel verwerker sigbaar nie.

Stelsel 2 en 3:

Vir gevalle waar die interne CAN eenhede van die verwerkers in stelsel 3 in die eksterne geheue spasie van die verwerker gekarteer is, is die verwerkerlas dieselfde as in die geval van die alleenstaande CAN eenhede, wat ook in die eksterne geheue spasie gekarteer is. Indien die interne CAN eenheid aangespreek word deur interne spesiale funksie registers (SFR's) en daar van die interne adres en databus gebruik gemaak word, is die toegang tot die CAN eenheid vinniger as in die geval met die eksterne CAN eenheid. Aangesien die verwerker toegang tot die CAN eenheid in hierdie geval vinniger is, is die verwerkerlas ook ligter.

Kommentaar:

In vergelyking met stelsel 1 is die verwerkerlas van stelsels 2 en 3 aansienlik hoër, weens die feit dat die verwerker die opstel van die CAN eenheid en die beheer daaroor moet behartig. Die eienskap wat egter die grootste invloed op die verwerkerlas het, is

die hoeveelheid data wat op 'n keer tussen die netwerkeenheid en die substelsel daaraan gekoppel oorgedra kan word. Die onderbrekingslas vir die oordrag van groot boodskappe is baie hoog vir stelsels 2 en 3 in vergelyking met stelsel 1. Vir byvoorbeeld die stuur van data kan die substelsel gekoppel aan stelsel 1 die hele boodskap aan die netwerkeenheid se geheue oordra en die boodskap kan uit geheue versend word. Vir stelsels 2 en 3 kan die substelsel slegs 8 datagrepe op 'n slag aan die netwerkeenheid oordra. Wanneer hierdie 8 grepe dan suksesvol versend is, genereer die netwerkeenheid 'n onderbreking wat aandui dat die transmissie afgehandel is en die substelsel kan die volgende 8 grepe data aan die netwerkeenheid oordra vir versending. Wanneer 'n boodskap in die geval van stelsel 2 en 3 ontvang word, word slegs 8 datagrepe op 'n slag deur die netwerkeenheid ontvang en hierdie 8 grepe moet dan aan die substelsel oorgedra word. Die substelsel word deur middel van 'n onderbreking in kennis gestel van die data wat gelees moet word.

6.6 Uitbreibaarheid:

Die uitbreibaarheid van die stelsel is baie belangrik aangesien die doel van die ondersoek is om 'n stelsel te ontwerp wat maklik aan 'n verskeidenheid stelsels gekoppel kan word.

Stelsel 1 en 2:

Die 2 stelsels kan maklik in die eksterne adres spasie van 'n verwerker karteer word. 'n Standaard drywer kan ook vir beide stelsels geskryf word in 'n hoë vlak taal soos C en slegs vertaal word vir 'n sekere teiken verwerker argitektuur. 'n Nuwe drywer hoef nie vir elke nuwe verwerker geskryf te word nie.

Stelsel 3:

Die CAN eenhede van verskillende verwerkers verskil in werking, met die gevolg dat 'n drywer wat geskryf is vir een tipe verwerker se CAN eenheid nie noodwendig sal werk vir 'n ander tipe verwerker se CAN eenheid nie. Die drywer kan miskien, met slegs 'n klein hoeveelheid veranderinge, vir verskillende tipe verwerkers werk, indien daar van verwerkers van dieselfde vervaardiger gebruik gemaak word. In die algemeen sal die drywer egter van nuuts af ontwikkel moet word en dit is nie 'n gewenste eienskap vir 'n generiese stelsel nie.

6.7 Keuse van Stelsel:

Aangesien die opstelling vir stelsels 2 en 3 baie dieselfde is, is dit die moeite werd om hierdie 2 stelsels teen mekaar op te weeg. Die 2 stelsels verskil nie veel in terme van koste en verwerkerlas nie. Stelsel 3 het 'n voordeel bo stelsel 2 in terme van 'n 1 minder komponent. Stelsel 2 het egter 'n groot voordeel bo stelsel 3 in terme van stelsel uitbreibaarheid. Die doel van die ontwerp is om 'n generiese eenheid daar te stel en die feit dat daar vir stelsel 2 'n generiese drywer geskryf kan word, maak hierdie stelsel meer ideaal vir implementasie as stelsel 3.

Die vraag is dus watter een van stelsel 1 of 2 die mees geskikte eienskappe het. Vir beide hierdie stelsels kan 'n generiese drywer ontwikkel word wat vir 'n verskeidenheid verwerkers stelsels gebruik kan word. Stelsel 1 se groot voordeel bo stelsel 2 is dat die verwerkerlas aansienlik laer is. Stelsel 1 se koste, komponent telling en stroomverbruik is egter baie hoër as in die geval met stelsel 2.

Die vraag wat beantwoord moet word is hoe belangrik is 'n lae verwerkerlas in terme van onderbrekings vanaf die netwerkeenheid. Die prys wat vir 'n lae verwerkerlas betaal word, indien stelsel 1 gekies word, is 'n hoër stroomverbruik, meer komponente en 'n hoër koste.

Aangesien daar aan 'n verskeidenheid verwerkers gekoppel word, is dit moeilik om die invloed wat die hoër verwerkerlas van stelsel 2 op die verwerker daaraan gekoppel sal hê vooraf te bepaal.

Beide stelsels het dus voordele sowel as nadele, maar nie 1 van die 2 stelsels is ideaal vir algemene data kommunikasiestelsel nie.

Hoofstuk 7: Gevolgtrekking

Werk behandel in hierdie tesis:

In hierdie tesis is daar gekyk na die huidige kommunikasie struktuur van SUNSAT 1. Tekortkominge van die huidige stelsel is uitgewys en daar is gekyk na die behoeftes vir 'n volgende generasie SUNSAT kommunikasie struktuur.

Binne hierdie raamwerk is die moontlikheid ondersoek om die CAN protokol te gebruik as deel van so 'n algemene kommunikasie struktuur. Drie verskillende opstellings is geïdentifiseer waardeur CAN in 'n laespoed kommunikasiestelsel gebruik kan word. Die ontwerp en werking van die verskillende stelsels is bespreek en die stelsels se eienskappe is vergelyk om 'n ideale stelsel te vind vir gebruik in 'n volgende generasie kommunikasiestelsel.

Kan CAN gebruik word in 'n algemene data kommunikasiestelsel ?

In terme van algemene data kommunikasie het CAN sy beperkinge ten opsigte van pakkie grootte en die hoë verwerkerlas as gevolg van die klein pakkie grootte. Indien 'n koste effektiewe oplossing geskep word met die minimum komponente en die minimum stroomverbruik ontstaan 'n hoë verwerkerlas by die stelsel verwerker gekoppel aan die netwerkeenheid. Indien 'n stelsel ontwikkel word met 'n minimum verwerkerlas verhoog die koste, stroomverbruik en aantal komponente benodig drasties.

CAN kan dus vir algemene data kommunikasie gebruik word, maar teen 'n prys. 'n Ideale implemantasie kan nie gevind word nie en die hoof rede hiervoor is die klein hoeveelheid data wat per boodskap versend kan word.

Verdere Werk:

Voordat enige spesifieke netwerkstelsel gekies word vir implementasie in 'n inter-satelliet kommunikasiestelsel moet die behoeftes van die stelsel eers duidelik daar gestel word.

Vrae soos die volgende moet beantwoord word:

- Hoeveel nodusse kom op die netwerk voor ?
- Wat is die verlangde tempo van data oordrag ?
- Is daar 'n sekere verlangde minimum of maksimum pakkie grootte ?
- Wat is die verwagte netwerk las ?
- Wat is die intydse behoeftes van die stelsels op die netwerk ?
- Word meer as een kommunikasiestelsel verlang ?
- Wat is die kragverbruik beperkings op die stelsel ?

Wanneer hierdie vrae beantwoord is en die behoefte vir 'n laespoed kommunikasiestelsel geïdentifiseer word, moet daar ook na ander moontlike tipe protokolle en netwerk opstellings behalwe CAN gekyk word. Hierdie verskillende stelsels moet dan met mekaar vergelyk word om 'n geskikte oplossing te vind.

Bronnelys:

[1] Lawrenz, W, ***CAN System Engineering – From Theory to Practical Applications***, Springer 1997

[2] Stallings, W. ***Data And Computer Communications*** **– Fifth Edition**, Prentice Hall Inc, 1997

[3] Badenhorst, P. ***The Development of a memory system for the second generation SUNSAT micro-satellite***, M-Tesis, 1997

[4] Goosen, N. ***Die ontwerp en evaluering van die sekondêre rekenaar in SUNSAT***, M-Tesis, 1996

[5] Gouws, H. ***Design of an on-board computer for the SUNSAT micro-satellite***, M-Tesis, 1996

[6] Du Toit, J. ***Computer System for the SUNSAT micro-satellite***, M-Tesis, 1994

[7] Mackay, S. ***PC Instrumentation for the 90s,*** Boston Technical Books, 1994

[8] Bosch, R. ***CAN Specification – Version 2***, Robert Bosch GmbH, 1991

[9] MacKenzie, S. ***The 8051 Microcontroller – Second Edition***, Prentice Hall Inc, 1992

[10] Beach, M. ***The C51 Primer***, Hitex (UK) Ltd, 1996

Bylaes:

1. CAN Beskrywing:

CAN is 'n asinchrone seriële bus stelsel wat ekstensief in verspreide intydse stelsels gebruik word. Twee bus logiese vlakke kom voor naamlik 'n resessiewe toestand (gewoonlik deur 'n logiese een voorgestel word) en 'n dominante toestand (gewoonlik logiese vlak 0). Solank geen bus nodus 'n dominante bis stuur nie, is die bus lyn in die resessiewe toestand, maar 'n dominante bis van enige nodus plaas die bus in 'n dominante toestand. Vir die CAN bus lyn moet daar dus 'n medium gekies word wat in staat is om twee moontlike bus toestande te versend naamlik resessiewe en dominant. Een van die maklikste en goedkoopste metodes is om gedraaide paar koperdraad te gebruik. Die gedraaide paar draad word aan 'n CAN sender/ontvanger verbind wat dan weer aan die CAN verwerker verbind is. Die maksimum bus spoed is 1Mbisse per sekonde, wat verkry kan word met 'n bus lengte van tot 40m. Aangesien die transmissie oor die CAN bus van differensiële seine gebruik maak is die transmissie redelik bestand teen EMI (Elektromagnetiese Inmenging).

Die binêre data word met behulp van die "NRZ" ("Non Return to Zero") skema enkodeer. Om die eksakte sinkronisasie van al die bus nodusse te verseker word bis inplasing gebruik.

Bus arbitrasie word gedoen deur middel van die "CSMA/CD" ("Carrier Sense Multiple Access Collision Detection with Non Destructive Arbitration"). Enige nodus wat data oor die netwerk wil stuur wag totdat die bus in die rus toestand is. Die nodus begin dan met die transmissie van 'n begin vlaggie gevolg deur die raam se identifiseerder. Indien meer as 1 nodus gelyktydig 'n boodskap wil versend, word die vernietiging van beide boodskappe verhoed deur van bis vlak arbitrasie gebruik te maak. Hierdie bis vlak arbitrasie sorg dat die boodskap met die hoogste prioriteit versend word. Die ander nodus bemerk dat 'n boodskap met 'n hoër prioriteit versend word en staak transmissie. Sodra die bus weer in die rus toestand is, kan hierdie nodus weereens probeer om sy boodskap te versend.

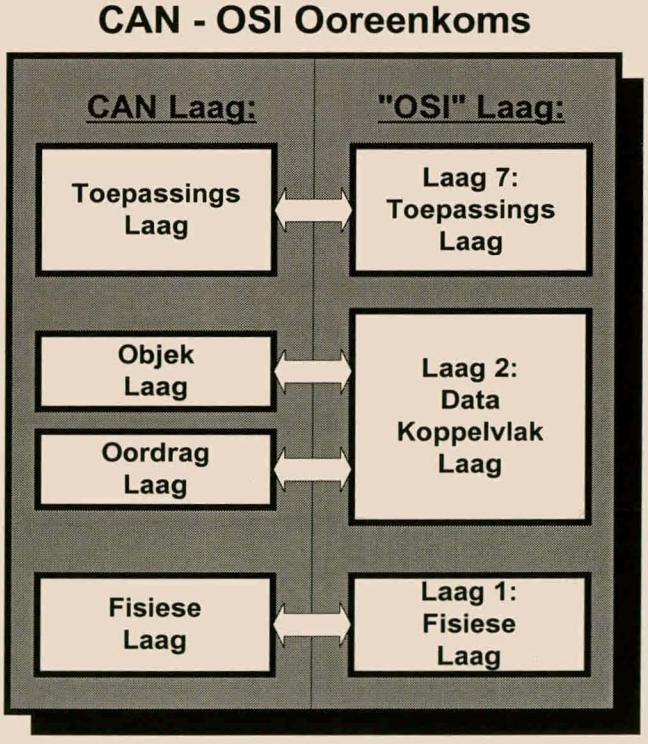
Met weergawe 2.0B van die CAN protokol is dit moontlik om rame met 'n 11 of 29 bis identifiseerder te versend oor die netwerk. Die identifiseerder dui ook die prioriteit van die boodskap aan.

CAN EIENSKAPPE:

- Boodskap prioriteits toekenning
- Boodskap versending kan aan sekere intydse vereistes voldoen
- Buigbare konfigurasie
- Multi-ontvanger boodskappe met tyd sinkronisasie
- Multimeester
- Foutdeteksie en aanduiding
- Automatiese hersending van foutiewe boodskappe sodra die bus weer in die Rus toestand is
- Onderskeiding tussen tydelike foute en permanente falings van nodusse en automatiese afskakeling van 'n defektiewe nodus

STRUKTUUR VAN CAN NODE:

Die funksionaliteit van die CAN kommunikasie argitektuur met verwysing na die "ISO/OSI" ("International Standardisation Organisation/ Open System Interconnect") model word in die onderstaande figuur getoon.



Figuur 23: CAN – OSI Ooreenkoms

FISIESE LAAG:

Hierdie laag spesifiseer hoe die seine oor die fisiese medium gestuur word. Dit spesifiseer ook die tipe medium wat gebruik word. Vir die CAN protokol word daar algemeen van differensiële kommunikasie oor gedraaide paar koperdraad gebruik gemaak.

OORDRAG LAAG:

Hierdie laag stel die kern van die CAN protokol voor. Dit is verantwoordelik vir sinkronisasie, bis tydsberekening, foutdeteksie en fout beperking.

OBJEK LAAG:

Hierdie laag is verantwoordelik vir boodskap filtering asook status en boodskap hantering.

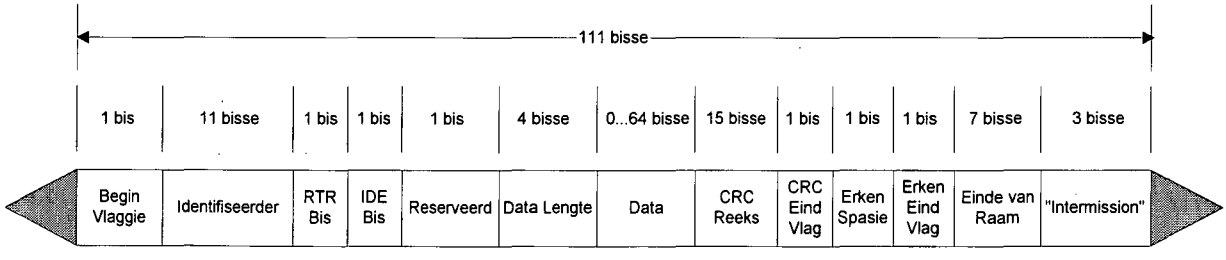
TOEPASSING LAAG:

Die toepassing laag stel die koppelvlak voor wat deur die toepassing gebruik word vir netwerk kommunikasie.

TIPE RAME:

STANDAARD CAN DATA RAAM:

'n Standaard CAN data raam word in figuur. 24 getoon.



Figuur 24: Standaard CAN Data Raam

'n Data raam word deur 'n node genereer wat data wil stuur. Die raam begin met 'n Begin vlaggie vir sinkronisasie met al die ander nodusse.

Die Begin vlaggie bis word gevolg deur die arbitrasie veld wat bestaan uit 12 bisse. Die arbitrasie veld bestaan uit 'n 11 bis identifiseerder en 'n RTR ("Remote Transmit Request"). Hierdie RTR bis word gebruik om 'n data raam te onderskei van 'n "Remote" raam.

Die volgende veld in die raam is die beheer veld wat bestaan uit 6 bisse. Die eerste bis van hierdie veld is die "IDE" ("Identifier Extension") bis en word gebruik om te onderskei tussen 'n standaard en uitgebreide raam. Die volgende bis is gereserveerd. Die volgende 4 bisse staan bekend as die Data Lengte en dui die aantal datagrepe aan wat in die boodskap voorkom. 'n Maksimum van 8 grepe kan in 'n raam versend word.

Die volgende veld is die data veld. Die hoeveelheid data wat in hierdie veld voorkom word bepaal deur die getal in die Data Lengte veld.

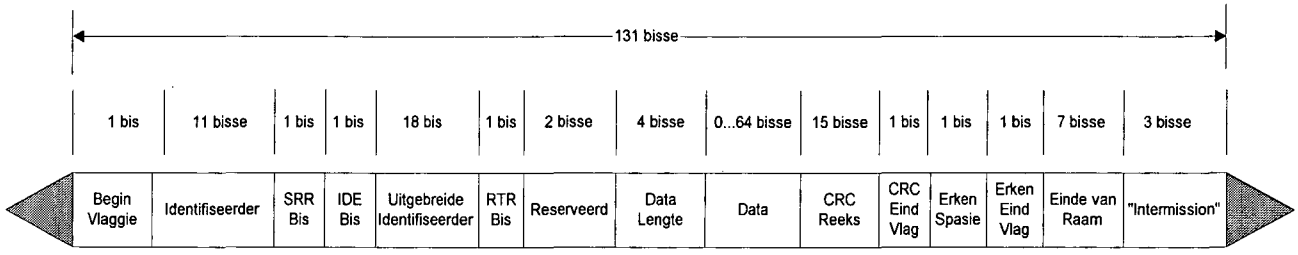
Die "CRC" veld volg en bestaan uit 15 bisse. Na die 15 "CRC" bisse volg 'n "CRC" eind vlaggie.

Die laaste veld is die Erken veld, wat 1 bis beslaan. Gedurende die versending van hierdie bis word 'n resessiewe bis versend. Enige nodus wat 'n fout vrye raam ontvang het, stuur dan 'n dominante bis uit (dit word gedoen ongeag of die nodus opgestel is om daardie spesifieke boodskap te ontvang of nie). Die CAN protokol maak dus van in bis terugvoering gebruik. Die Erken Eind Vlag sluit die Erken veld af.

Die raam word afgesluit deur 7 resessiewe bisse. Tussen enige 2 rame moet die bus in die resessiewe toestand bly vir minstens 3 bis periodes, wat bekend staan as die "Intermission". Indien daar geen ander node is wat 'n raam wil stuur na die "Intermission" nie bly die bus in die rus toestand.

UITGEBREIDE CAN RAAM:

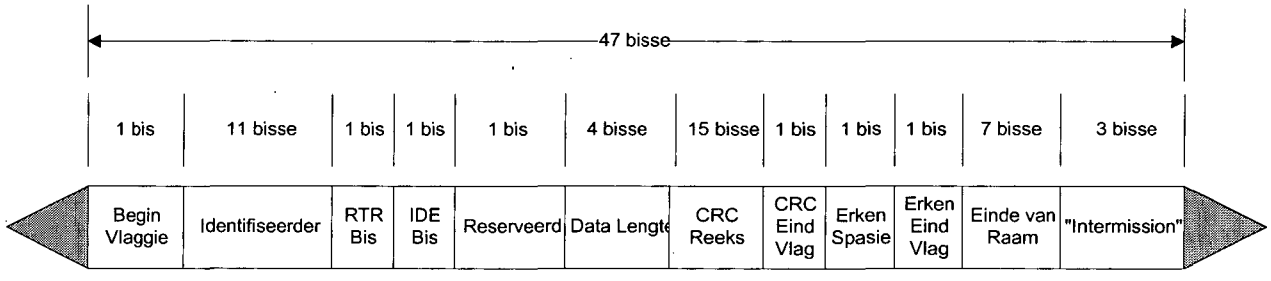
Dit is ook moontlik om van 'n 29 bis identifiseerder gebruik te maak. So 'n raam staan bekend as 'n uitgebreide data raam. Figuur 25 toon so 'n raam.



Figuur 25: Uitgebreide CAN Raam

"REMOTE" RAME:

Data transmissie vind gewoonlik plaas deurdat 'n node data na 'n ander node stuur m.b.v. 'n data raam. Dit is egter ook moontlik om 'n ander node te vra vir data. Dit word gedoen m.b.v. 'n "Remote" raam, wat in Figuur 26 getoon word.



Figuur 26: CAN "Remote" Raam

Daar is 2 verskille tussen 'n data en 'n "Remote" raam. Eerstens is die "RTR" bis in die resessiewe toestand en tweedens is daar geen data veld nie.

FOUT RAME:

'n Fout Raam word uitgestuur deur enige nodus wat 'n bus fout opmerk. 'n Fout raam bestaan uit 2 velde, 'n Fout vlag gevolg deur 'n Fout "Delimiter" veld wat bestaan uit 8 resessiewe bisse.

Indien 'n "fout aktiewe" nodus 'n bus fout opmerk, onderbreek die node die transmissie van die huidige boodskap deur 'n aktiewe fout vlaggie te genereer. Hierdie aktiewe fout vlaggie bestaan uit 6 agtereenvolgende dominante bisse. Hierdie bis sekwensie oortree die bis inplasing reël. Alle ander stasies sien die bis inplasing fout raak en genereer dan ook Fout Rame. Die Fout Raam veld bestaan dus uit tussen 6 en 12 agtereenvolgende dominante

bisse, wat genereer word deur een of meer nodusse. Die Fout "Delimiter" veld sluit die fout raam af. Na afloop van die Fout Raam keer die bus aktiwiteit na normaal terug en die onderbreekte nodus kan die raam weer stuur.

Indien 'n "passiewe fout" nodus 'n bus fout opmerk, stuur dit 'n "passiewe fout" vlaggie gevolg deur die Fout "Delimiter" veld. Die "passiewe fout" vlag bestaan uit 6 agtereenvolgende resessiewe bisse gevolg deur die Fout "Delimiter" veld wat bestaan uit 8 resessiewe bisse. Die raam bestaan dus uit 14 agtereenvolgende resessiewe bisse.

TIPE FOUTE:

1. **"SOT" Fout.**

2. **"Acknowledge" Fout:**

'n "Acknowledge" Fout kom voor indien geen nodus 'n gestuurde raam korrek ontvang nie.

3. **Vorm Fout:**

Indien die nodus wat 'n raam stuur 'n dominante bis opmerk in een van die "End of Frame" "Interframe Space", "Acknowledge Delimiter" of die "SOT Delimiter" opmerk het 'n vorm fout voorgekom en 'n fout raam word genereer.

4. **Bis Fout:**

'n Bis fout kom voor indien die gestuurde bis waarde nie ooreenstem met die waarde wat van die bus af gelees word nie. Indien 'n resessiewe bis gestuur word en dominante bis gelees word, in die Arbitrasie of "Acknowledge" tydgleuf, word 'n fout boodskap nie genereer nie.

Indien 'n fout boodskap genereer word, moet die raam weer gestuur word.

OORLAAI RAAM:

'n Oorlaai raam het dieselfde formaat as 'n Fout raam. So 'n raam kan egter net genereer word tydens die "Interframe Space". 'n Oorlaai raam kan deur 'n nodus genereer word a.g.v. twee toestande:

1. Indien 'n node 'n dominante bis waarneem gedurende die "Interframe Space".

2. Indien a.g.v. interne toestande 'n nodus nie gereed is om 'n volgende boodskap te begin ontvang nie.

CAN BEHEERDER:

'n Tipiese CAN beheerder bestaan uit die volgende hoof komponente:

Protokol Beheerder:

Die protokol beheer is verantwoordelik vir alle boodskappe wat deur die CAN bus gestuur of ontvang word. Dit sluit take soos sinkronisasie, fout hantering, arbitrasie parallel na serie en serie na parallel omskakeling in.

Boodskap Filters:

Die hardeware filter is verantwoordelik vir die filtering van alle boodskappe. Die doel van hierdie filter is om die las op die mikroverwerker te verlig deur slegs boodskappe te aanvaar wat vir daardie spesifieke nodus bedoel is.

Boodskap Buffers:

Die stuur en ontvang buffers word gebruik om boodskappe tydelik te stoor.

Verwerker Koppelvlak:

Beide alleenstaande en geïntegreerde CAN eenhede het 'n verwerker koppelvlak eenheid waardeur die verwerker beheer uitoefen oor die CAN eenheid.

2. Foutdeteksie en korreksie:

'n Fout in 'n digitale stelsel kom voor indien die data verander van die korrekte waarde na enige ander waarde. Hierdie foute kan veroorsaak word deur 'n wye verskeidenheid bronne.

R.W. Hamming het in 1950 'n metode ontwikkel vir die konstruksie van kodes met die eienskappe dat dit enkel bis foute kan korrigeer en dubbel bis foute kan identifiseer. 'n Kodewoord word uitgewerk vir elke data greep wat in die geheue gestoor moet word en die kodewoord word saam met die data in die geheue gestoor. Die kodewoord word bepaal deur middel van eksklusiewe-of somme. Vir 'n data wydte van 8 bisse is die kodewoord se wydte 5 bisse. Sodra die data en die kodewoord dan weer uit die geheue gelees word, word daar met behulp van die kodewoord bepaal of daar enige foute in die data voorkom. Enkel bis foute word gekorrigeer en dubbel bis foute word geïdentifiseer.

3. Berekening van foutdeteksie en korreksie stelsel stroomverbruik:

Die interne stroomverbruik is die som van die bystands en die operasionele stroomverbruik. Die bystands stroomverbruik (Icc bystand) word in die ALTERA datavel aangetoon as 5 mA. Die aktiewe stroomverbruik (Icc aktief) word as volg bereken:

$$Icc\ aktief = K \times fmax \times Npers \times 10^{-6}$$

K ➔ Konstante verkry uit datavel tabel.
Fmax ➔ Maksimum operasionele frekwensie in MHz.
Npers ➔ Persentasie van totale aantal logiese selle wat elke kloksiklus verander.

Die waardes vir bogenoemde veranderlikes is vir hierdie geval as volg:

K = 88;
Fmax = 12;
Npers = 0.125*55 = 6.875;

Met bostaande waardes is Icc aktief = 7.26 mA.
Die totale interne stroomverbruik (Iint) is dus Icc aktief plus Icc bystand:
Iint = (7.26 + 5) mA = 12.26 mA

4. SYNOPSYS DESIGNWARE SOT Stelsel:

Die polinoom wat gebruik word om die SOT te genereer is die volgende:

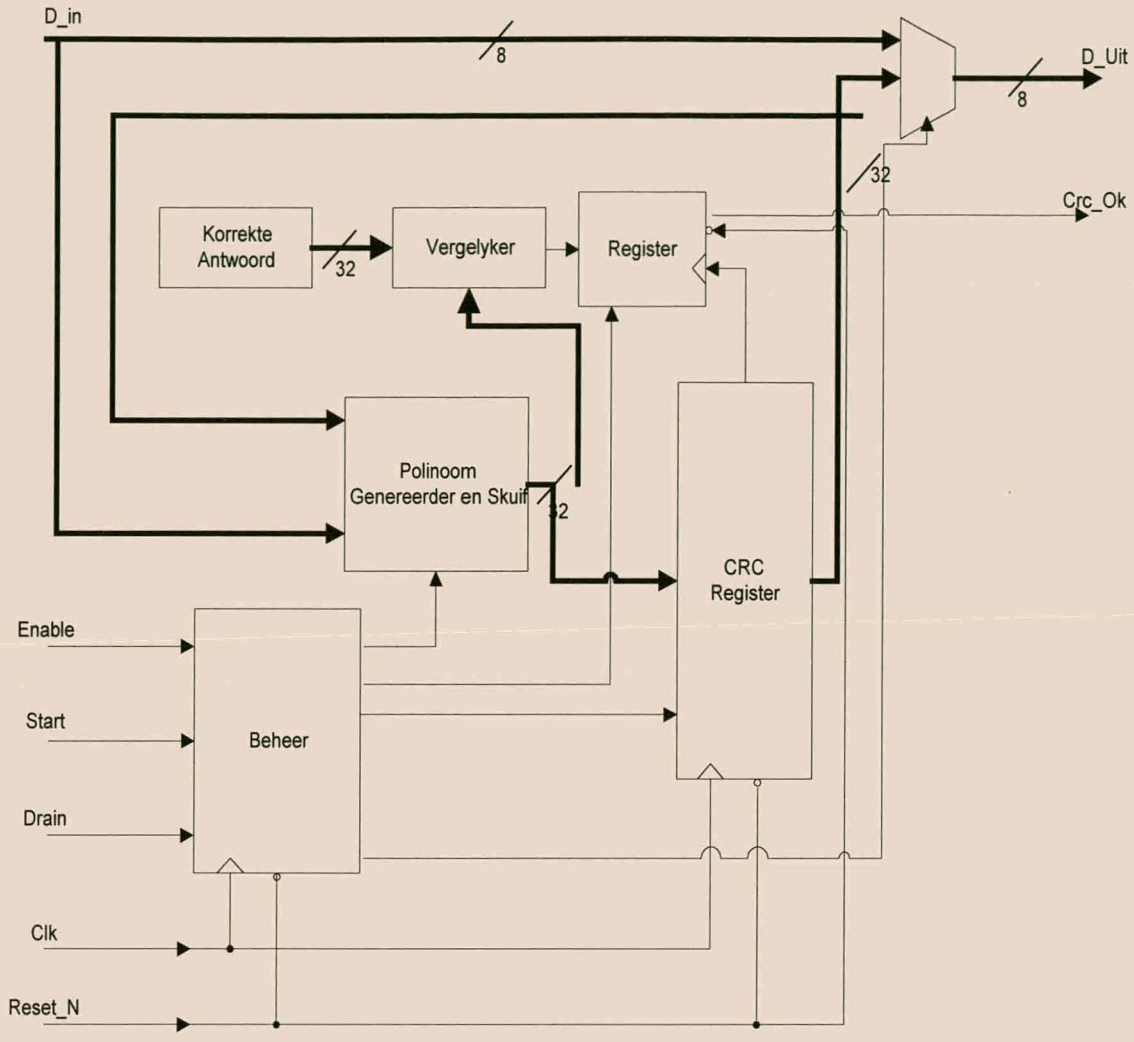
$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

Indien die SOT funksie in genereer modus geplaas word, word al die SOT skuif register se bisse geïnisialiseer na 1 en die resulterende SOT word geïnverteer voordat dit vrygestel word.

In die toets modus word die SOT skuif register bisse weereens geïnisialiseer na een en sodra die hele data stroom sowel as die SOT stroom deur die stelsel gestuur is, sal die res van die bewerking as volg lyk indien daar geen data korruptering voorgekom het nie:

$X^{31} + X^{30} + X^{26} + X^{25} + X^{24} + X^{18} + X^{15} + X^{14} + X^{12} + X^{11} + X^{10} + X^8 + X^6 + X^5 + X^4 + X^3 + X + 1$

Die stelsel blokdiagram word in die onderstaande figuur getoon:



Figuur 27: Stelsel Blokdiagram van SOT Eenheid

Pen Beskrywing:

Pen Naam:	Grootte:	Tipe:	Funksie:
D_In	8	Ingang	Ingang Data
Clk	1	Ingang	Stelsel Klok
Reset_N	1	Ingang	Sinkrone herstel, Aktief laag
Enable	1	Ingang	Aktiveer pen vir alle bewerkings, Aktief hoog
Start	1	Ingang	Begin bewerking in die SOT register
Drain	1	Ingang	Tap die waarde uit die SOT register
D_Out	8	Uitgang	Uitgang Data
CRC_Ok	1	Uitgang	Dui korrekte SOT waarde aan.

Tabel 16: Pen beskrywing van SOT Eenheid

Werking:

Die "enable" lyn dien as die selekteer sein vir die stelsel. Indien hierdie lyn laag is bly die stelsel in sy huidige toestand.

Sodra 'n opgaande rand van die klok voorkom en die "start" lyn is hoog, word daar met die berekening van die SOT begin sodra die volgende opgaande klok rand voorkom. Sodra die "start" lyn laag gaan word die 32 bis skuif register se bisse almal na 1 gestel. Die stelsel voer dan 'n SOT bewerking uit op die ingang data "D_in" vir elke kloksiklus wat die "enable" lyn hoog is.

Wanneer die hele data stroom deur die stelsel gestuur is moet die 32 bis SOT waarde vanaf die stelsel verkry word om dit agter aan die blok data te plaas. Dit word gedoen deur die "Drain" ingang hoog te neem. Tydens die volgende vier kloksiklusse word die 32 bis SOT waarde dan by die uitgang datapoort "D_out" uitgelees, met die mees belangrikste 8 bisse van die 32 bis getal eerste en die 8 mins belangrikste bisse laaste.

Indien die "Drain" ingang laag gehou word voer die stelsel 'n SOT toets uit op die datastroom wat by die ingang databis aangelê word. Indien 'n korrekte SOT bemerk word word die "Crc_Ok" vlaggie gestel.

5. Berekening van SOT stroomverbruik:

Die interne stroomverbruik is die som van die bystands en die operasionele stroomverbruik. Die bystands stroomverbruik (Icc bystand) word in die ALTERA datavel aangetoon as 5 mA. Die aktiewe stroomverbruik (Icc aktief) word as volg bereken:

Icc aktief = K x fmax x Npers x 10^{-6}

K ➔ Konstante verkry uit datavel tabel.
Fmax ➔ Maksimum operasionele frekwensie in MHz.
Npers ➔ Persentasie van totale aantal logiese selle wat elke kloksiklus verander.

Die waardes vir bogenoemde veranderlikes is vir hierdie geval as volg:

K = 88;
Fmax = 12;
Npers = 0.125*119 = 14.875;

Met bostaande waardes is Icc aktief = 15.708 mA.
Die totale interne stroomverbruik (Iint) is dus Icc aktief plus Icc bystand:
Iint = (15.708 + 5) mA = 20.708 mA

6. Berekening van totale FPGA stroomverbruik:

Die interne stroomverbruik is die som van die bystands en die operasionele stroomverbruik. Die bystands stroomverbruik (Icc bystand) word in die ALTERA datavel aangetoon as 5 mA. Die aktiewe stroomverbruik (Icc aktief) word as volg bereken:

$$Icc\ aktief = K \times fmax \times Npers \times 10^{-6}$$

K ➔ Konstante verkry uit datavel tabel.
Fmax ➔ Maksimum operasionele frekwensie in MHz.
Npers ➔ Persentasie van totale aantal logiese selle wat elke kloksiklus verander.

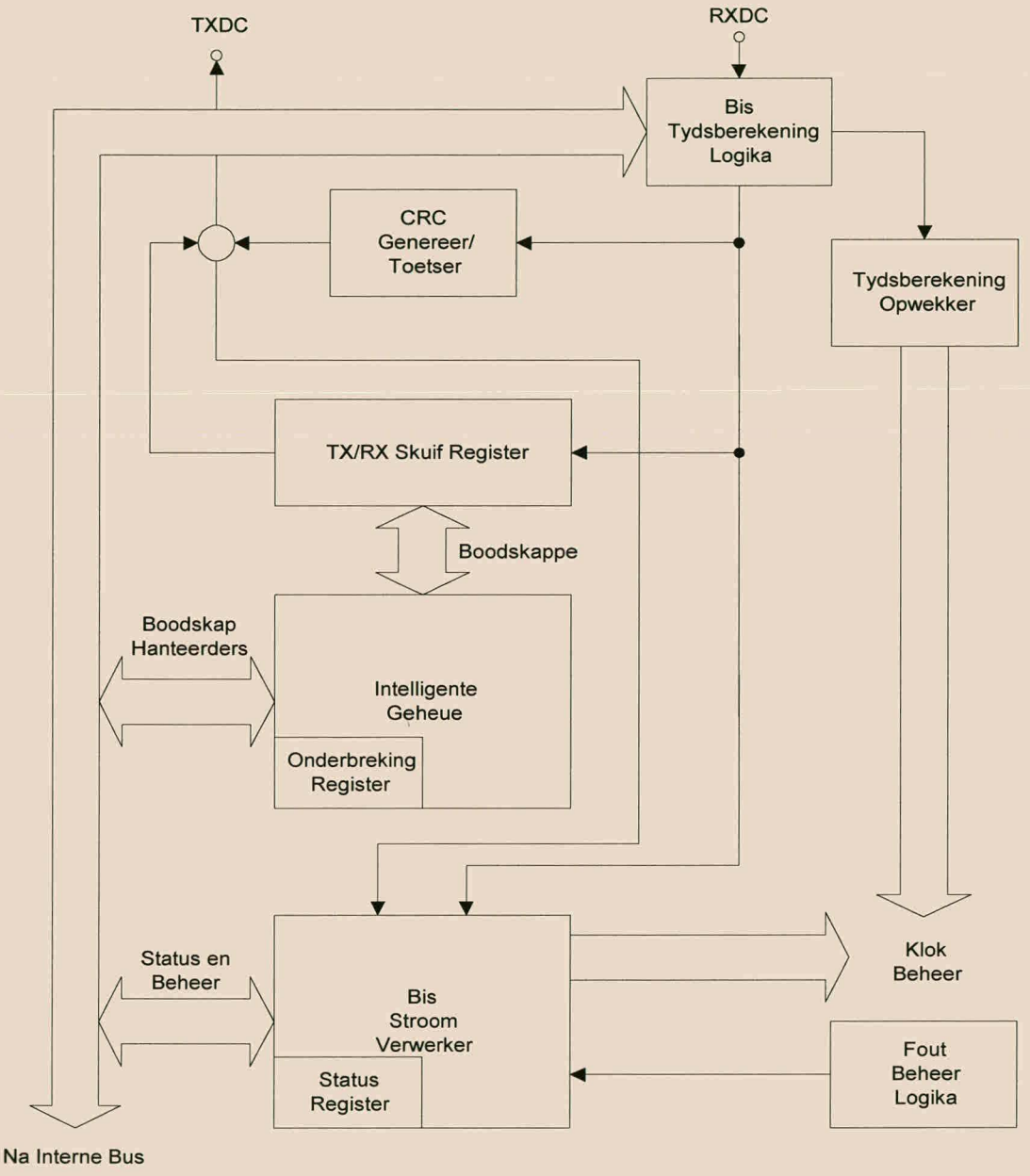
Die waardes vir bogenoemde veranderlikes is vir hierdie geval as volg:

K = 88;
Fmax = 12;
Npers = 0.125*970 = 121.25;

Met bostaande waardes is Icc aktief = 128.04 mA.
Die totale interne stroomverbruik (Iint) is dus Icc aktief plus Icc bystand:
Iint = (128.04 + 5) mA = 133.04 mA

7. Infineon C505C CAN eenheid:

'n Blokdiagram voorstelling van die CAN beheerder word in figuur 28 getoon:



Figuur 28: C505C CAN Eenheid

Can Beheerder Eenhede:

Die CAN eenheid bestaan uit 15 verskillende boodskap objekte wat elk bestaan uit 'n maksimum van 8 datagrepe. Elke boodskap objek kan afsonderlik gekonfigureer word deur 'n stel beheer registers. Elke boodskap objek het ook 'n stel status registers, sowel as stuur en ontvang onderbrekings. Elke objek kan opgestel word as 'n ontvang of 'n stuur boodskap deur middel van die beheer registers. Die CAN beheerder

ondersteun die standaard sowel as die uitgebreide formaat van die CAN protokol. 'n Globale boodskap filter bestaan vir 14 van die 15 boodskap objekte. Die laaste boodskap objek het sy boodskap filter.

Die hoof eenhede van die CAN eenheid is die volgende:

1. TX/RX Skuif Register:

Hierdie register behartig die serie/ parallel en parallel/serie omskakeling van die boodskappe wat oor die netwerk ontvang of versend word.

2. Bis Stroom Verwerker:

Die bus stroom verwerker beheer die data strome wat tussen die verskillende eenhede van die CAN eenheid vervoer word. Die bis stroom verwerker hanteer ook die automatiese retransmissie van die boodskappe wat gekorrupteer is as gevolg van ruis op die bus lyn.

3. Siklusse Oortolligheids Toets Register:

Hierdie eenheid is verantwoordelik vir die toets van die integriteit van die raam wat versend en ontvang word. Dit word gedoen deur middel van die bekende siklusse oortolligheids toets metode.

4. Fout Beheer Logika:

Hierdie eenheid bestaan uit 'n stuur sowel as 'n ontvang fout teller. Hierdie tellers word geïnkrementeer deur bevele vanaf die bis stroom verwerker sodra 'n stuur of ontvang fout voorkom. Die CAN eenheid verkeer in een van drie moontlike toestande na gelang van die fout tellers. Die drie toestande is 'n passiewe fout toestand, 'n aktiewe fout toestand en 'n bus af toestand.

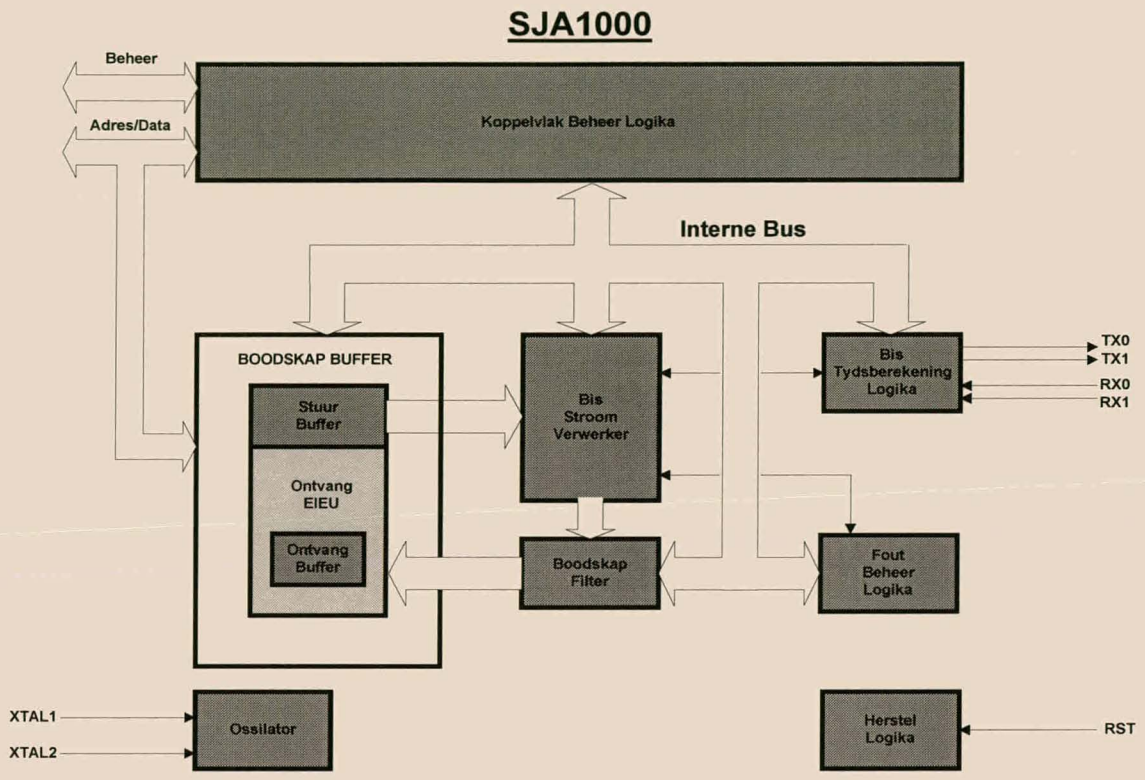
5. Bis Tydsberekening Logika:

Hierdie eenheid monitor die ontvang lyn vanaf die CAN sender/ontvanger en hanteer die sinkronisasie van die ontvanger op die bus.

6. Intelligente Geheue:

Die intelligente geheue dien as stoorplek vir die 8 datagrepe van die 15 verskillende boodskap objekte.

8. SJA1000 CAN eenheid:



Figuur 29: SJA1000 CAN Eenheid

Die hoof eenhede van die CAN eenheid is die volgende:

1. Bis Stroom Verwerker:

Die bus stroom verwerker beheer die data strome wat tussen die verskillende eenhede van die CAN eenheid vervoer word. Die bis stroom verwerker hanteer ook foutdeteksie en die automatiese retransmissie van die van boodskappe wat gekorrupteer is as gevolg van ruis op die bus lyn.

2. Fout Beheer Logika:

Hierdie eenheid bestaan uit 'n stuur sowel as 'n ontvang fout teller. Hierdie tellers word geïnkrementeer deur bevele vanaf die bis stroom verwerker sodra 'n stuur of ontvang fout voorkom. Die CAN eenheid verkeer in een van drie moontlike toestande na gelang van die fout tellers. Die drie toestande is 'n passiewe fout toestand, 'n aktiewe fout toestand en 'n bus af toestand.

3. Bis Tydsberekening Logika:

Hierdie eenheid monitor die ontvang lyn vanaf die CAN sender/ontvanger en hanteer die sinkronisasie van die ontvanger op die bus.

4. Koppelvlak Beheer Logika:

Hierdie eenheid hanteer die koppelvlak met die verwerker. Dit ontvang en interpreteer bevele vanaf die verwerker, hanteer die adressering van die CAN registers en lewer status inligting aan die verwerker wat aan die stelsel gekoppel is.

5. Stuur Buffer:

Die stuur buffer dien as koppelvlak tussen die verwerker en die Bis Stroom Verwerker. Die buffer word geskryf deur die verwerker en gelees deur die Bis Stroom Verwerker. Die stuur buffer is 13 grepe groot.

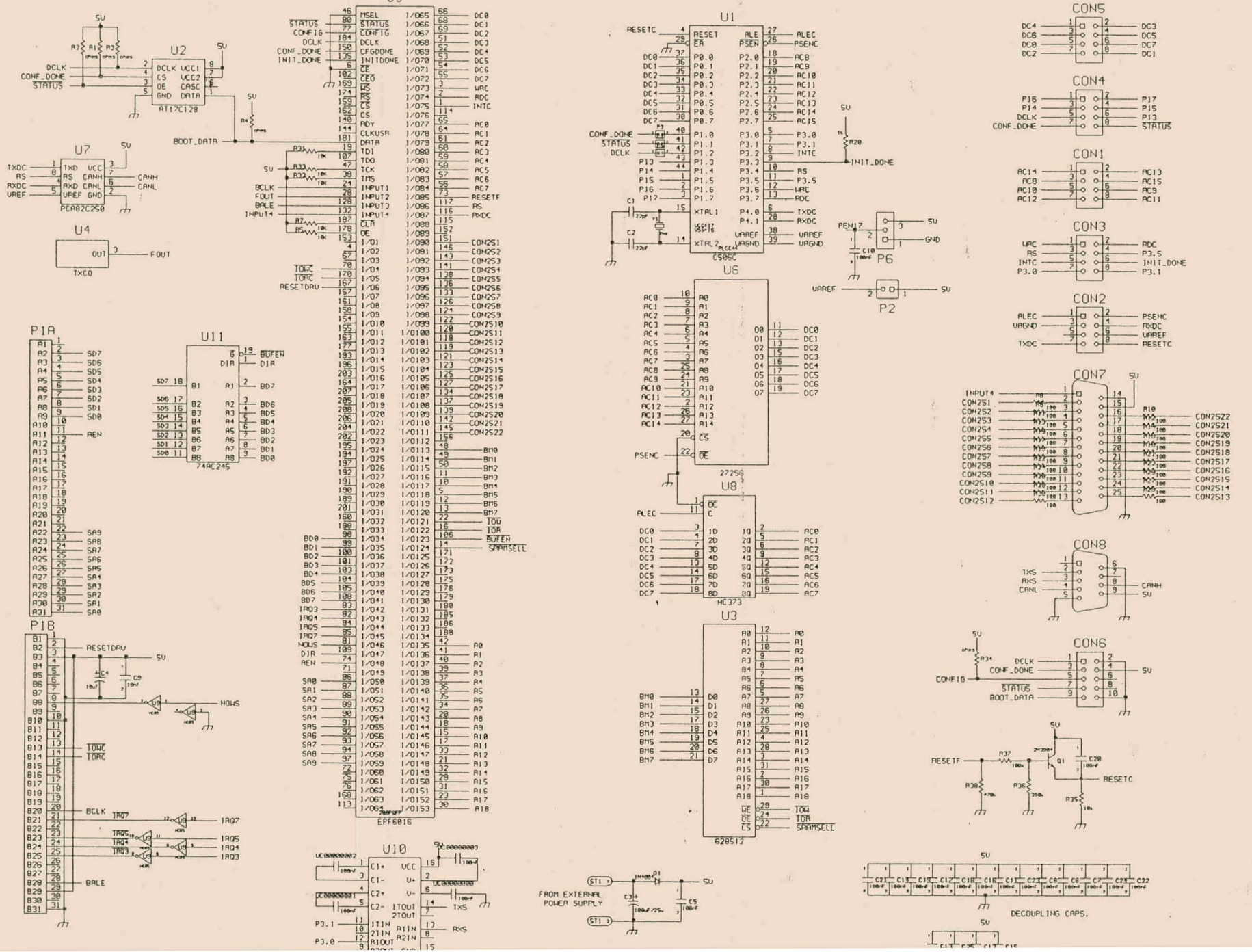
6. Ontvang Buffer:

Die ontvang buffer dek die eerste 13 grepe van die ontvang EIEU (Eerste In Eerste Uit) buffer en dien as tydelike stoorplek vir ontvangde boodskappe. Die verwerker kan die ontvangde boodskap uit die ontvang buffer lees. Die EIEU buffer is 64 grepe groot wat tot gevolg het dat meet as 1 ontvangde boodskap op 'n keer in die geheue gestoor kan word.

7. Boodskap Filter:

Die boodskap filter vergelyk die identifiseerder van ontvangde boodskappe met die waarde in die boodskap filter register om te bepaal of die boodskap in die ontvang EIEU buffer gestoor moet word.

9. Stelsel 1 Stroombaandiagram:



10. VHDL Kode:

```
------------------------------------------------------------------------
-- Fileame : board.vhd
-- Author  : Hugo Steyn
-- Date    : 03-09-98
-- Target Device : 17C256 for EPF6016
-- Description :
-- Notes :
------------------------------------------------------------------------
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
entity board is
    Port (
                AEN             :       In              std_logic;
                SubBale         :       In              std_logic;
                CLK             :       In              std_logic;
                RS              :       In              std_logic;
                RXDC            :       In              std_logic;
                SubIOWC         :       In              std_logic;
                CanIOWC         :       In              std_logic;
                SubIORC         :       In              std_logic;
                CanIORC         :       In              std_logic;
                HERSTEL         :       In              std_logic;
                SubSa           :       In              std_logic_vector(9 downto 0);
                CanSa           :       In              std_logic_vector(7 downto 0);

                INTSub          :       InOut   std_logic;
                INTCan          :       InOut   std_logic;
                NOE             :       InOut   std_logic;
                NWE             :       InOut   std_logic;
                NCS             :       InOut   std_logic;
                MemAdrs         :       InOut   std_logic_vector(18 downto 0);
                MemData         :       InOut   std_logic_vector(7 downto 0);
                SubBD           :       InOut   std_logic_vector(7 downto 0);
                CanBD           :       InOut   std_logic_vector(7 downto 0);

                BUFEN           :       Out             std_logic;
                BUFDIR          :       Out             std_logic;
                IRQ3            :       Out             std_logic;
                IRQ4            :       Out             std_logic;
                IRQ5            :       Out             std_logic;
                IRQ7            :       Out             std_logic;
                NOWS            :       Out             std_logic);
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
end board;
architecture BEHAVIORAL of board is
type WhatAction is (GetSize7to0,GetAdressHigh,GetAdressLow,GetBlock,GetDestination,GetData,GiveSize7to0,
        GiveData,GiveSource,GiveRegA,GiveRegB,CheckCrc,ClearRegA,SetRegB,GiveLast,NOP);

type MemorySignals is (Reset,ReadSub1,ReadSub2,WriteSub1,WriteSub2,ReadCan1,ReadCan2,WriteCan1,WriteCan2);

signal CrcTest          : std_logic_vector(7 downto 0);
signal Counter          : integer range 0 to 3;         -- Indicates which CRC is clocked out

signal Counter2         : unsigned(1 downto 0);         -- Indicates which Crc Value is given to RAM
signal ActivateWrite    : std_logic;                    -- ActivateWrites Counter 2
signal ActivateRead     : std_logic;                    -- ActivateWrites Counter 2
signal TAct             : std_logic;                    -- ActivateWrites Counter 2
signal PsMS,NsMS        : MemorySignals;                -- State Machine for accessing RAM
signal SubAction        : WhatAction;                   -- Indicates what Subsystem does
signal WriteSubData     : std_logic_vector(7 downto 0); -- Register storing data from and to Subsystem
signal ReadSubData      : std_logic_vector(7 downto 0); -- Register storing data from and to Subsystem
signal TempSubData      : std_logic_vector(7 downto 0); -- Value given to SubBD on SubBale event
signal TempCanData      : std_logic_vector(7 downto 0); -- Value given to CanBD on NCSOut event
signal DoSubRead        : boolean;                      -- Indicates if Subsystem memory read is taking place
signal DoSubWrite       : boolean;                      -- Indicates if Subsystem memory write is taking place
signal SubAdrs          : unsigned(10 downto 0);        -- Send memory address
signal CanAdrs          : unsigned(10 downto 0);        -- Send memory address
signal SubBusy          : std_logic;                    -- Indicates subsystem memory access is in progress
```

```
signal NCSin                : std_logic;                        -- Driver for Memory CS signal
signal NCSout               : std_logic;                        -- Driver for Memory CS signal
signal CanAction            : WhatAction;                       -- Indicates what Can system does
signal CanData              : std_logic_vector(7 downto 0);     -- Register storing data from and to Cansystem
signal DoCanRead            : boolean;                          -- Indicates if Can system memory read is taking place
signal DoCanWrite           : boolean;                          -- Indicates if Can system memory read is taking place
signal Reset_N              : std_logic;                        --\
signal T1                   : std_logic;                        -- |
signal Enable               : std_logic;                        -- |
signal Start                : std_logic;                        -- |
signal Drain                : std_logic;                        -- |
signal Byte_time            : std_logic;                        -- |
signal Draining             : std_logic;                        -- |
signal Accumulating         : std_logic;                        -- \ Crc
signal Crc_Ok               : std_logic;                        -- / Signals
signal D_in                 : std_logic_vector(7 downto 0);     -- |
signal D_Out                : std_logic_vector(7 downto 0);     -- |
signal CrcActive            : std_logic_vector(7 downto 0);     -- |
signal Crc0                 : std_logic_vector(7 downto 0);     -- |
signal Crc1                 : std_logic_vector(7 downto 0);     -- |
signal Crc2                 : std_logic_vector(7 downto 0);     -- |
signal Crc3                 : std_logic_vector(7 downto 0);     -- |
signal SizeA1               : std_logic_vector(7 downto 0);
signal SizeA2               : std_logic_vector(7 downto 0);
signal SizeA3               : std_logic_vector(7 downto 0);
signal SizeA4               : std_logic_vector(7 downto 0);
signal SizeB1               : std_logic_vector(7 downto 0);
signal SizeB2               : std_logic_vector(7 downto 0);
signal SizeB3               : std_logic_vector(7 downto 0);
signal SizeB4               : std_logic_vector(7 downto 0);
signal AdressA1             : std_logic_vector(7 downto 0);
signal AdressA2             : std_logic_vector(7 downto 0);
signal AdressA3             : std_logic_vector(7 downto 0);
signal AdressA4             : std_logic_vector(7 downto 0);
signal AdressB1             : std_logic_vector(7 downto 0);
signal AdressB2             : std_logic_vector(7 downto 0);
signal AdressB3             : std_logic_vector(7 downto 0);
signal AdressB4             : std_logic_vector(7 downto 0);
signal ClearA               : std_logic_vector(3 downto 0);
signal ClearB               : std_logic_vector(3 downto 0);
signal RegA1                : std_logic;
signal RegA2                : std_logic;
signal RegA3                : std_logic;
signal RegA4                : std_logic;
signal RegB1                : std_logic;
signal RegB2                : std_logic;
signal RegB3                : std_logic;
signal RegB4                : std_logic;
signal SubBlock             : std_logic_vector(1 downto 0);
signal CanBlock             : std_logic_vector(1 downto 0);
signal SubEndAdrs           : unsigned(10 downto 0);
signal SubSendCurrentSize   : unsigned(7 downto 0);
signal SubReceiveCurrentSize        : unsigned(7 downto 0);
signal SubRWClk                     : std_logic;
signal LastData                     : std_logic_vector(7 downto 0);
-------------------------------------------------------------------
COMPONENT GLOBAL
  PORT (a_in : IN STD_LOGIC;
      a_out: OUT STD_LOGIC);
END COMPONENT;
COMPONENT crc32_DW04_crc32_3_3_0
          PORT(   D_In                :       In          std_logic_vector(7 downto 0);
                  Clk                 :       In          std_logic;
                  Reset_N             :       In          std_logic;
                  Enable              :       In          std_logic;
                  Byte_Time           :       In          std_logic;
                  Start               :       In          std_logic;
                  Drain               :       In          std_logic;
                  D_Out               :       Out         std_logic_vector(7 downto 0);
                  Accumulating        :       Out         std_logic;
                  Draining            :       Out         std_logic;
                  Crc_Ok              :       Out         std_logic);
END COMPONENT;
begin
U1: crc32_DW04_crc32_3_3_0
```

```
        PORT MAP (      D_in                    =>      D_in,
                        Clk                     =>      Clk,
                        Reset_N                 =>      Reset_N,
                        Enable                  =>      Enable,
                        Byte_Time               =>      Byte_Time,
                        Start                   =>      Start,
                        Drain                   =>      Drain,
                        D_Out                   =>      D_Out,
                        Accumulating            =>      Accumulating,
                        Draining                =>      Draining,
                        Crc_Ok                  =>      Crc_Ok);
U2: GLOBAL
        PORT MAP (a_in    => NCSin,
                  a_out   => NCSout);
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
SubRWClk        <= SubIORC AND SubIOWC;
Enable          <= T1 OR Start OR Draining;
IRQ3            <= '0';
IRQ4            <= '0';
IRQ5            <= '0';
IRQ7            <= '0';
NOWS            <= '0';
-------------------------------------------------------------------------------------------
----------------------------AdressA1Drv------------------------------------------------
--      AdressLowA1 from Subsystem given to Can Controller              --
-------------------------------------------------------------------------------------------
AdressA1Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 Begin
  if (SubIOWC = '0') and (SubAction = GetDestination) then
     Case SubBlock is
        when "00" =>
           AdressA1 <= SubBd;
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------------------------
----------------------------AdressLowDrv-----------------------------------------------
--      AdressLowA2 from Subsystem given to Can Controller              --
-------------------------------------------------------------------------------------------
AdressLowDrv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
  if (SubIOWC = '0') and (SubAction = GetDestination) then
     Case SubBlock is
        when "01" =>
           AdressA2 <= SubBd;
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------------------------
----------------------------AdressA3Drv------------------------------------------------
--      AdressLowA3 from Subsystem given to Can Controller              --
-------------------------------------------------------------------------------------------
AdressA3Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
  if (SubIOWC = '0') and (SubAction = GetDestination) then
     Case SubBlock is
        when "10" =>
           AdressA3 <= SubBd;
        when Others =>
      end Case;
  end if;
 end process;
-------------------------------------------------------------------------------------------
----------------------------AdressA4Drv------------------------------------------------
--      AdressLowA4 from Subsystem given to Can Controller              --
-------------------------------------------------------------------------------------------
AdressA4Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
  if (SubIOWC = '0') and (SubAction = GetDestination) then
     Case SubBlock is
        when "11" =>
```

```
            AdressA4 <= SubBd;
          when Others =>
       end Case;
    end if;
 end process;
-------------------------------------------------------------------------------
----------------------------AdressB1-------------------------------------------
--          AdressB1 from Can Controller given to Subsystem                  --
-------------------------------------------------------------------------------
AdressB1Drv:process(CanIOWC,CanBD,CanAction,CanBlock)
 begin
   if (CanIOWC = '0') and (CanAction = GetDestination) then
      Case CanBlock is
         when "00" =>
            AdressB1 <= CanBd;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------------
----------------------------AdressB2-------------------------------------------
--          AdressB2 from Can Controller given to Subsystem                  --
-------------------------------------------------------------------------------
AdressB2Drv:process(CanIOWC,CanBD,CanAction,CanBlock)
 begin
   if (CanIOWC = '0') and (CanAction = GetDestination) then
      Case CanBlock is
         when "01" =>
            AdressB2 <= CanBd;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------------
----------------------------AdressB3-------------------------------------------
--          AdressB3 from Can Controller given to Subsystem                  --
-------------------------------------------------------------------------------
AdressB3Drv:process(CanIOWC,CanBD,CanAction,CanBlock)
 begin
   if (CanIOWC = '0') and (CanAction = GetDestination) then
      Case CanBlock is
         when "10" =>
            AdressB3 <= CanBd;
         when Others =>
      end Case;
            end if;
 end process;
-------------------------------------------------------------------------------
----------------------------AdressB4-------------------------------------------
--          AdressHighB4 from Can Controller given to Subsystem              --
-------------------------------------------------------------------------------
AdressHighB4Drv:process(CanIOWC,CanBD,CanAction,CanBlock)
 begin
   if (CanIOWC = '0') and (CanAction = GetDestination) then
      Case CanBlock is
         when "11" =>
            AdressB4 <= CanBd;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------------
----------------------------SubActionDrv---------------------------------------
-------------------------------------------------------------------------------
SubActionDrv:process (HERSTEL,SubBale,SubSa,AEN)
 begin
   if SubBale ' event and SubBale = '0' then
      if ((SubSA(9 downto 4) = "110000") and (AEN='0')) then
         Case SubSa (3 downto 0) is
            when "0000" =>
               SubAction <= GetAdressHigh;                    --      300
            when "0001" =>
               SubAction <= GetAdressLow;                     --      301
            when "0010" =>
               SubAction <= GetBlock;                         --      302
```

```
            when "0011" =>
               SubAction <= GetSize7to0;                    --      303
            when "0100" =>
               SubAction <= GetDestination;                 --      304
            when "0101" =>
               SubAction <= GetData;                        --      305
            when "0110" =>
               SubAction <= GiveSize7to0;                   --      306
            when "0111" =>
               SubAction <= GiveData;                       --      307
            when "1000" =>
               SubAction <= GiveSource;                     --      308
            when "1001" =>
               SubAction <= CheckCrc;                       --      309
            when "1010" =>
               SubAction <= GiveRegA;                       --      30A
            when "1011" =>
               SubAction <= GiveRegB;                       --      30B
            when "1100" =>
               SubAction <= GiveLast;                       --      30C
            when Others =>
               SubAction <= NOP;
         end case;
      else
         SubAction <= NOP;
      end if;
   end if;
end process;
-------------------------------------------------------------------------------
----------------------------BufEnDrv-------------------------------------------
-------------------------------------------------------------------------------
BufEnDrv:process (HERSTEL,SubBale,SubSa,AEN)
 begin
  if (HERSTEL = '1') then
     BufEn <= '1';
  else
     if SubBale ' event and SubBale = '0' then
        if ((SubSa(9 downto 4) = "110000") and (AEN='0')) then
           case SubSa (3 downto 0) is
              when   "0000" | "0001" | "0010" | "0011" | "0100" | "0101"  | "0110" | "0111" | "1000" | "1001" | "1010" | "1011" | "1100"  =>
                 BufEn <= '0';
              when others =>
                 BufEn <= '1';
            end case;
        else
           BufEn <= '1';
        end if;
     end if;
  end if;
 end process;
-------------------------------------------------------------------------------
----------------------------BufDirDrv------------------------------------------
-------------------------------------------------------------------------------
BufDir <= NOT SubIORC;
-------------------------------------------------------------------------------
----------------------------SubBDDrv-------------------------------------------
---------------- Driver for the Sub System Data Bus---------------------------
-------------------------------------------------------------------------------
SubBDDrv:process(SubIORC)
 begin
  if SubIORC = '0' then
     Case SubAction is
        when GiveSource =>
           Case SubBlock is
              when "00" =>
                 SubBd <= AdressB1;
              when "01" =>
                 SubBd <= AdressB2;
              when "10" =>
                 SubBd <= AdressB3;
              when "11" =>
                 SubBd <= AdressB4;
              when Others =>
           end Case;
        when GiveData =>
```

```
            SubBD <= TempSubData;
          when CheckCrc =>
              SubBD <= CrcTest;
          when GiveSize7to0 =>
            Case SubBlock is
              when "00" =>
                 SubBd <= SizeB1;
              when "01" =>
                 SubBd <= SizeB2;
              when "10" =>
                 SubBd <= SizeB3;
              when "11" =>
                 SubBd <= SizeB4;
              when Others =>
            end Case;
          when GiveRegA =>
            SubBd(7 downto 4) <= "0000";
            SubBd(3 downto 0) <= RegA4 & RegA3 & RegA2 & RegA1;
          when GiveRegB =>
            SubBd(7 downto 4) <= "0000";
            SubBd(3 downto 0) <= RegB4 & RegB3 & RegB2 & RegB1;
          when GiveLast =>
            SubBd <= LastData;
          when Others =>
            SubBD <= "ZZZZZZZZ";
      end Case;
   else
       SubBD <= "ZZZZZZZZ";
   end if;
end process;
-------------------------------------------------------------------------------
------------------------------INTSubDrv----------------------------------------
--              Driver for the Can Controller Interrupt                      --
-------------------------------------------------------------------------------
INTSubDrv:process(SubAction,Herstel,CanIOWC,CanAction,SubAction)
begin
  if (Herstel = '1') or (SubAction = GiveSource) then
    INTSub <= '0';
  elsif CanIOWC ' event and CanIOWC = '1' then
      Case CanAction is
         when SetRegB =>
            INTSub <= '1';
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------------
------------------------------INTCanDrv----------------------------------------
--              Driver for the Can Controller Interrupt                      --
-------------------------------------------------------------------------------
INTCanDrv:process(CanAction,CanIORC,Herstel,SubAdrs,SubEndAdrs,NCSOut)
VARIABLE Sum : unsigned(10 downto 0);
begin
  Sum := SubEndAdrs + 4;
  if (Herstel = '1') or ((CanAction = GiveRegA) and (CanIORC = '0')) then
     INTCan <= '1';
  elsif NCSOut ' event and NCSOut  = '0' then
     if (SubAdrs = Sum ) then
        INTCan <= '0';
     end if;
  end if;
 end process;
-------------------------------------------------------------------------------
------------------------------SubAdrsDrv---------------------------------------
--              Address driver for subsystem memory access.                  --
-------------------------------------------------------------------------------
SubAdrsDrv:process(PsMs,SubIOWC,SubAction,NCSout,SubBd)
 begin
   if (SubIOWC = '0') and (SubAction = GetAdressHigh) then
      SubAdrs(10 downto 8) <= unsigned(SubBd(2 downto 0));
   elsif (SubIOWC = '0') and (SubAction = GetAdressLow) then
      SubAdrs(7 downto 0) <= unsigned(SubBd);
   elsif NCSout ' event and NCSout = '1' then
      case PsMs is
         when WriteSub2 | ReadSub2 =>
```

```
            SubAdrs <= SubAdrs + 1;
          when Others =>
       end case;
    end if;
 end process;
-------------------------------------------------------------------------------
------------------------------CanAdrsDrv---------------------------------------
--              Address driver for Can system memory access.   --
-------------------------------------------------------------------------------
CanAdrsDrv:process(PsMs,CanIOWC,CanAction,NCSout,CanBd)
 begin
  if (CanIOWC = '0') and (CanAction = GetAdressLow) then
     CanAdrs(10 downto 8) <= "000";
     CanAdrs(7 downto 0) <= unsigned(CanBd);
  elsif NCSout ' event and NCSout = '1' then
     case PsMs is
        when WriteCan2 | ReadCan2 =>
           CanAdrs <= CanAdrs + 1;
        when Others =>
     end case;
  end if;
 end process;
-------------------------------------------------------------------------------------------------
------------------------------MemAdrsDrv-----------------------------------------------------------
--Defines if address for subsystem or adres for can is allocated to the memory  --
-------------------------------------------------------------------------------------------------
MemAdrsDrv:process(NCS,PsMs,SubAdrs,CanAdrs)
 begin
  MemAdrs(18 downto 11) <= "00000000";
  if NCS = '1' then
     Case PsMs is
        when WriteSub1 | WriteSub2 | ReadSub1 | ReadSub2=>
           MemAdrs(10 downto 0) <= conv_std_logic_vector(SubAdrs,11);
        when others =>
           MemAdrs(10 downto 0) <= conv_std_logic_vector(CanAdrs,11);
     end Case;
  end if;
end process;
-------------------------------------------------------------------------------------------------
------------------------------DoSubRead----------------------------------------------------------
--                        Indicates if Subsystem memory read is taking place        --
-------------------------------------------------------------------------------------------------
DoSubReadDrv:process(Herstel,ActivateRead,SubAction,SubIORC,PsMs,Clk)
 begin
  if (Clk = '1') and((Herstel = '1') or (PsMs= ReadSub2)) then
     DoSubRead <= FALSE;
  elsif ActivateRead = '1' then
     DoSubRead <= True;
  elsif SubIORC ' event and SubIORC = '0' then
     case SubAction is
        when GiveData =>
           DoSubRead <= True;
        when Others =>
           DoSubRead <= False;
     end case;
   end if;
 end process;
-------------------------------------------------------------------------------------------------
------------------------------DoSubWrite---------------------------------------------------------
--                        Indicates if Subsystem memory write is taking place        --
-------------------------------------------------------------------------------------------------
DoSubWriteDrv:process(Herstel,ActivateWrite,SubAction,SubIOWC,PsMs,Clk)
 begin
  if (Herstel = '1') or ((Clk ='1') and  (PsMs= WriteSub2)) then
     DoSubWrite <= FALSE;
  elsif ActivateWrite = '1' then
     DoSubWrite <= True;
  elsif SubIOWC ' event and SubIOWC = '1' then
     case SubAction is
        when GetData  =>
           DoSubWrite <= True;
        when Others =>
           DoSubWrite <= False;
     end case;
   end if;
```

```
  end process;
-------------------------------------------------------------------------------
--              Updating of Memory Signals State Machine              --
-------------------------------------------------------------------------------
PsMsDrv:process(Clk,NsMs)
 begin
  if Clk ' event and Clk = '0' then
     PsMs <= NsMs;
  end if;
 end process;
-------------------------------------------------------------------------------
--              Next State for Memory Signals State Machine           --
-------------------------------------------------------------------------------
NsMsDrv:process(PsMs,DoSubRead,DoSubWrite,DoCanRead,DoCanWrite)
 begin
  case PsMs is
     when Reset =>
        if DoCanRead then
           NsMs <= ReadCan1;
        elsif DoCanWrite then
           NsMs <= WriteCan1;
        elsif DoSubRead then
           NsMs <= ReadSub1;
        elsif DoSubWrite then
           NsMs <= WriteSub1;
        else
           NsMs <= Reset;
        end if;
     when ReadSub1 =>
        NsMs <= ReadSub2;
     when ReadSub2 =>
        NsMs <= Reset;
     when WriteSub1 =>
        NsMs <= WriteSub2;
     when WriteSub2 =>
        NsMs <= Reset;
     when ReadCan1 =>
        NsMs <= ReadCan2;
     when ReadCan2 =>
        NsMs <= Reset;
     when WriteCan1 =>
        NsMs <= WriteCan2;
     when WriteCan2 =>
        NsMs <= Reset;
  end case;
 end process;
-------------------------------------------------------------------------------
-----------------------------SubBusyDrv----------------------------------------
--          Indicates when subsystem memory access is in progress     --
-------------------------------------------------------------------------------
SubBusyDrv:process(Clk,PsMs)
begin
 if Clk = '1' then
    Case PsMs is
       when ReadSub1 | ReadSub2 | WriteSub1 | WriteSub2 =>
          SubBusy <= '1';
       when Others =>
          SubBusy <= '0';
     end Case;
 end if;
end process;
-------------------------------------------------------------------------------
-------------------------------ReadSubBDDrv-----------------------------------
-------------------------------------------------------------------------------
ReadSubBDDrv:process(NCS)
 begin
 if  (NCS = '0')  then
    case PsMs is
       when ReadSub2 =>
          ReadSubData <= MemData;
       when others =>
    end case;
 end if;
end process;
-------------------------------------------------------------------------------
```

```
----------------------------------CrcActiveDrv-----------------------------------------------
--                                Register storing data from and to Subsystem          --
-----------------------------------------------------------------------------------------
CrcActiveDrv:process(ActivateWrite,NCS,PsMs)
 begin
  Case Counter2 is
     when "00" =>
        CrcActive <= Crc0;
     when "01" =>
        CrcActive <= Crc1;
     when "10" =>
        CrcActive <= Crc2;
     when "11" =>
        CrcActive <= Crc3;
     when Others =>
   end case;
end process;
-----------------------------------------------------------------------------------------
-------------------------WriteSubDataDrv-------------------------
--    Register storing data from and to Subsystem          --
-----------------------------------------------------------------------------------------
WriteSubDataDrv:process(SubIOWC,ActivateWrite)
 begin
  if SubIOWC ' event and SubIOWC = '1' then
     if (SubAction = GetData) then
        WriteSubData <= SubBD;
     end if;
 end if;
end process;
-----------------------------------------------------------------------------------------
---------------------------TempSubDataDrv-----------------------------
--    TempSubData stores value given to SubBD on SubBale event        --
-----------------------------------------------------------------------------------------
TempSubDataDrv:process(SubBale,ReadSubData)
 begin
  if SubBale ' event and SubBale = '0' then
     TempSubData <= ReadSubData;
  end if;
 end process;
-----------------------------------------------------------------------------------------
-----------------------------MemDataDrv-----------------------------------
--             Driver for the memory Data Bus                    --
-----------------------------------------------------------------------------------------
MemDataDrv:process(NCS,PsMs)
 begin
  if (NCS = '1') then
     case PsMs is
        when WriteSub1 | WriteSub2 =>
           if ActivateWrite = '1' then
              MemData <= CrcActive;
           else
              MemData <= WriteSubData;
           end if;
         when WriteCan1 | WriteCan2 =>
            Memdata <= CanData;
         when others =>
            MemData <= "ZZZZZZZZ";
     end case;
  end if;
 end process;
-----------------------------------------------------------------------------------------
-----------------------------NCSinDrv-----------------------------------
--             Driver for the memory CS signal                   --
-----------------------------------------------------------------------------------------
NCSinDrv:process(HERSTEL,PsMs,Clk)
 begin
  if Clk ' event and Clk = '1' then
     if HERSTEL = '1' then
        NCSin <= '1';
     else
        case PsMs is
           when Reset | ReadSub2 |  WriteSub2 | ReadCan2 | WriteCan2 =>
              NCSin <= '1';
           when ReadSub1 |  WriteSub1 | ReadCan1 | WriteCan1 =>
              NCSin <= '0';
```

```
        end case;
      end if;
    end if;
  end process;
-------------------------------------------------------------------------
-----------------------------NCSDrv--------------------------------------
--            Driver for the memory CS signal                          --
-------------------------------------------------------------------------
NCSDrv:process(HERSTEL,PsMs,Clk)
 begin
   if Clk ' event and Clk = '1' then
      if HERSTEL = '1' then
         NCS <= '1';
      else
         case PsMs is
            when Reset | ReadSub2 |  WriteSub2 | ReadCan2 | WriteCan2 =>
               NCS <= '1';
            when ReadSub1 |  WriteSub1 | ReadCan1 | WriteCan1 =>
               NCS <= '0';
         end case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
-----------------------------NOEDrv--------------------------------------
--            Driver for the memory Rd signal                          --
-------------------------------------------------------------------------
NOEDrv:process(HERSTEL,PsMs,Clk)
 begin
   if Clk ' event and Clk = '1' then
      if HERSTEL = '1' then
         NOE <= '1';
      else
         case PsMs is
            when ReadSub1 | ReadCan1 =>
               NOE <= '0';
            when Others =>
               NOE <= '1';
         end case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
-----------------------------NWEDrv--------------------------------------
--            Driver for the memory Wr signal                          --
-------------------------------------------------------------------------
NWEDrv:process(HERSTEL,PsMs,Clk)
 begin
   if Clk ' event and Clk = '1' then
      if HERSTEL = '1' then
         NWE <= '1';
      else
         case PsMs is
            when WriteSub1 | WriteCan1 =>
               NWE <= '0';
            when Others =>
               NWE <= '1';
         end case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
---------------------------CanActionDrv----------------------------------
-------------------------------------------------------------------------
CanActionDrv:process (HERSTEL,CanSA)
 begin
   if HERSTEL = '1' then
      CanAction <= NOP;
   else
      if (CanSA(7 downto 4) = "0000") then
         case CanSA (3 downto 0) is
            when "0000" =>
               CanAction <= GetAdressHigh;                    --        0
            when "0001" =>
               CanAction <= GetAdressLow;                     --        1
```

```
            when "0010" =>
               CanAction <= GetBlock;                    --      2
            when "0011" =>
               CanAction <= GetSize7to0;                 --      3
            when "0100" =>
               CanAction <= GetDestination;              --      4
            when "0101" =>
               CanAction <= GetData;                     --      5
            when "0110" =>
               CanAction <= GiveSize7to0;                --      6
            when "0111" =>
               CanAction <= GiveData;                    --      7
            when "1000" =>
               CanAction <= GiveSource;                  --      8
            when "1001" =>
               CanAction <= GiveRegA;                    --      9
            when "1010" =>
               CanAction <= GiveRegB;                    --      A
            when "1011" =>
               CanAction <= ClearRegA;                   --      B
            when "1100" =>
               CanAction <= SetRegB;                     --      C
            when others =>
               CanAction <= NOP;
         end case;
      else
         CanAction <= NOP;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
----------------------------------CanDataDrv-----------------------------
--  Register storing data from and to Can system          --
-------------------------------------------------------------------------
CanDataDrv:process(Memdata,PsMs,NCS,CanAction,CanBD)
 begin
   if (CanIOWC = '0') and (CanAction = GetData) then
      CanData <= CanBD;
   elsif  NCS = '0' then
      Case PsMs is
         when ReadCan2 =>
            CanData <= MemData;
               when Others =>
      end case;
   end if;
 end process;
-------------------------------------------------------------------------
-----------------------------CanBDDrv------------------------------------
---------------- Driver for the Can Data Bus-----------------------------
-------------------------------------------------------------------------
CanBDDrv:process(CanIORC)
 begin
   if CanIORC = '0' then
      Case CanAction is
         when GiveSource  =>
            Case CanBlock is
               when "00" =>
                  CanBd <= AdressA1;
               when "01" =>
                  CanBd <= AdressA2;
               when "10" =>
                  CanBd <= AdressA3;
               when "11" =>
                  CanBd <= AdressA4;
               when Others =>
            end Case;
         when GiveData =>                                  --8
            CanBD <= TempCanData;
         when GiveSize7to0 =>
            Case CanBlock is
               when "00" =>
                  CanBd <= SizeA1;
               when "01" =>
                  CanBd <= SizeA2;
               when "10" =>
```

```
                CanBd <= SizeA3;
              when "11" =>
                CanBd <= SizeA4;
              when Others =>
            end Case;
          when GiveRegA =>
            CanBD(7 downto 4) <= "0000";
            CanBD(3 downto 0) <= RegA4 & RegA3 & RegA2 & RegA1;
          when GiveRegB =>
            CanBD(7 downto 4) <= "0000";
            CanBD(3 downto 0) <= RegB4 & RegB3 & RegB2 & RegB1;
          when Others =>
            CanBD <= "ZZZZZZZZ";
      end Case;
   else
      CanBD <= "ZZZZZZZZ";
   end if;
  end process;
-------------------------------------------------------------------------
------------------------------TempCanDataDrv-----------------------------
--            TempCanData stores value given to CanData                --
-------------------------------------------------------------------------
TempCanDataDrv:process(NCSout,CanData)
 begin
   if NCSout ' event and NCSout = '1' then
      TempCanData <= CanData;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------DoCanRead----------------------------------
-------------------------------------------------------------------------
DoCanReadDrv:process(Herstel,CanAction,CanIORC,PsMs,Clk)
  begin
    if (Clk = '1') and ((Herstel = '1') or (PsMs= ReadCan2)) then
       DoCanRead <= FALSE;
    elsif CanIORC ' event and CanIORC = '0' then
       case CanAction is
          when GiveData =>
             DoCanRead <= True;
          when Others =>
             DoCanRead <= False;
       end case;
    end if;
   end process;
-------------------------------------------------------------------------
------------------------------DoCanWrite---------------------------------
-------------------------------------------------------------------------
DoCanWriteDrv:process(CanAction,CanIOWC,PsMs,Clk)
  begin
    if ((Clk = '1') and (PsMs = WriteCan2)) THEN
       DoCanWrite <= FALSE;
    elsif CanIOWC ' event and CanIOWC = '1' then
       case CanAction is
          when GetData  =>
             DoCanWrite <= True;
          when Others =>
             DoCanWrite <= False;
       end case;
    end if;
   end process;
-------------------------------------------------------------------------
----------------------------------Reset_NDrv-----------------------------
-------------------------------------------------------------------------
Reset_NDrv:process(SubIORC,SubAction)
 begin
   if SubIORC ' event  and SubIORC = '1' then
      case SubAction is
         when CheckCrc =>
            Reset_N <= '0';
         when others =>
            Reset_N <= '1';
      end case;
   end if;
 end process;
-------------------------------------------------------------------------
```

```
----------------------------------T1Drv-----------------------------------
--------------------------------------------------------------------------
T1Drv:process(Clk)
begin
  if Clk ' event and Clk = '1' then
     if (NCS = '0') and (SubBusy = '1') then
        T1 <= '1';
     else
        T1 <= '0';
     end if;
  end if;
end process;
--------------------------------------------------------------------------
----------------------------------StartDrv---------------------------------
--------------------------------------------------------------------------
StartDrv:process(SubIOWC,Herstel,SubAction)
 begin
  if ((SubIOWC = '0') and (SubAction = GetDestination)) or ((SubIORC = '0') and (SubAction = GiveSource)) then
     Start <= '1';
  else
     Start <= '0';
  end if;
 end process;
--------------------------------------------------------------------------
----------------------------------DrainDrv---------------------------------
--------------------------------------------------------------------------
DrainDrv:process(Clk,SubAdrs,DoSubWrite,SubEndAdrs)
 begin
  if Clk ' event and Clk = '1' then
     if ((SubAdrs = SubEndAdrs) and (DoSubWrite)) then
        Drain <= '1';
     else
        Drain <= '0';
     end if;
  end if;
end process;
--------------------------------------------------------------------------
----------------------------------Byte_timeDrv-----------------------------
--------------------------------------------------------------------------
Byte_time <= '1';
--------------------------------------------------------------------------
----------------------------------CounterDrv-------------------------------
--------------------------------------------------------------------------
CounterDrv:process(Draining,Clk,Counter)
 begin
  if Clk ' event and Clk = '1' then
     if Draining = '1' then
        Counter <= Counter + 1;
     else
        Counter <= 0;
     end if;
  end if;
 end process;
--------------------------------------------------------------------------
----------------------------------CRC0Drv----------------------------------
--------------------------------------------------------------------------
Crc0Drv:process(Counter,Draining,Clk,D_Out)
 begin
  if Clk ' event and Clk = '1' then
     if Draining = '1' then
        case Counter is
           when 0 =>
              CRC0 <= D_Out;
           when others =>
        end case;
     end if;
  end if;
 end process;
--------------------------------------------------------------------------
----------------------------------CRC1Drv----------------------------------
--------------------------------------------------------------------------
Crc1Drv:process(Counter,Draining,Clk,D_Out)
 begin
  if Clk ' event and Clk = '1' then
    if Draining = '1' then
```

```
        case Counter is
          when 1 =>
            CRC1 <= D_Out;
          when others =>
        end case;
      end if;
    end if;
  end process;
-------------------------------------------------------------------------
----------------------------------CRC2Drv--------------------------------
-------------------------------------------------------------------------
Crc2Drv:process(Counter,Draining,Clk,D_Out)
 begin
  if Clk ' event and Clk = '1' then
     if Draining = '1' then
        case Counter is
          when 2 =>
            CRC2 <= D_Out;
          when others =>
        end case;
     end if;
  end if;
 end process;
-------------------------------------------------------------------------
----------------------------------CRC3Drv--------------------------------
-------------------------------------------------------------------------
Crc3Drv:process(Counter,Draining,Clk,D_Out)
 begin
  if Clk ' event and Clk = '1' then
     if Draining = '1' then
        case Counter is
          when 3 =>
            CRC3 <= D_Out;
          when others =>
        end case;
     end if;
  end if;
 end process;
-------------------------------------------------------------------------
----------------------------------D_inDrv--------------------------------
-------------------------------------------------------------------------
D_inDrv:process(PsMs,WriteSubData,MemData,NCSout)
 begin
  if NCSout ' event and NCSout = '1' then
     Case PsMs is
        when ReadSub1 | ReadSub2 =>
          D_in <= MemData;
        when Others =>
          D_in <= WriteSubData;
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
----------------------------------CrcTestDrv-----------------------------
-------------------------------------------------------------------------
CrcTest0Drv:process(Clk,MemAdrs,Crc_Ok,PsMs)
 begin
  CrcTest(0) <= Crc_Ok;
  CrcTest(7 downto 1) <= "0000000";
end process;
-------------------------------------------------------------------------
---------------------------------- SizeA1-------------------------------
-------------------------------------------------------------------------
SizeA1Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
  if (SubIOWC = '0') then
     Case SubAction is
        when GetSize7to0 =>
          case SubBlock is
            when "00" =>
              SizeA1 <= SubBd;
            when Others =>
          end case;
        when Others =>
```

```
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------ SizeA2-----------------------------------
-------------------------------------------------------------------------
SizeA2Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
   if (SubIOWC = '0') then
      Case SubAction is
         when GetSize7to0 =>
            case SubBlock is
               when "01" =>
                  SizeA2 <= SubBd;
               when Others =>
            end case;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------ SizeA3-----------------------------------
-------------------------------------------------------------------------
SizeA3Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
   if (SubIOWC = '0') then
      Case SubAction is
         when GetSize7to0 =>
            case SubBlock is
               when "10" =>
                  SizeA3 <= SubBd;
               when Others =>
            end case;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------ SizeA4-----------------------------------
-------------------------------------------------------------------------
SizeA4Drv:process(SubIOWC,SubBd,SubAction,SubBlock)
 begin
   if (SubIOWC = '0') then
      Case SubAction is
         when GetSize7to0 =>
            case SubBlock is
               when "11" =>
                  SizeA4 <= SubBd;
               when Others =>
            end case;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------ SizeB1-----------------------------------
-------------------------------------------------------------------------
SizeB1Drv:process(CanIOWC,CanBd,CanIOWC)
 begin
   if (CanIOWC = '0') then
      Case CanAction is
         when GetSize7to0 =>
            case CanBlock is
               when "00" =>
                  SizeB1 <= CanBd;
               when Others =>
            end case;
         when Others =>
      end Case;
   end if;
 end process;
-------------------------------------------------------------------------
------------------------------ SizeB2-----------------------------------
-------------------------------------------------------------------------
SizeB2Drv:process(CanIOWC,CanBd,CanIOWC)
```

```
 begin
  if (CanIOWC = '0') then
     Case CanAction is
        when GetSize7to0 =>
           case CanBlock is
              when "01" =>
                 SizeB2 <= CanBd;
              when Others =>
           end case;
        when Others =>
     end Case;
  end if;
 end process;
--------------------------------------------------------------------
---------------------------- SizeB3----------------------------
--------------------------------------------------------------------
SizeB3Drv:process(CanIOWC,CanBd,CanIOWC)
 begin
  if (CanIOWC = '0') then
     Case CanAction is
        when GetSize7to0 =>
           case CanBlock is
              when "10" =>
                 SizeB3 <= CanBd;
              when Others =>
           end case;
        when Others =>
     end Case;
   end if;
 end process;
--------------------------------------------------------------------
---------------------------- SizeB4----------------------------
--------------------------------------------------------------------
SizeB4Drv:process(CanIOWC,CanBd,CanIOWC)
 begin
  if (CanIOWC = '0') then
     Case CanAction is
        when GetSize7to0 =>
           case CanBlock is
              when "11" =>
                 SizeB4 <= CanBd;
              when Others =>
           end case;
        when Others =>
     end Case;
  end if;
 end process;
--------------------------------------------------------------------
---------------------------- ClearA----------------------------
--------------------------------------------------------------------
ClearADrv:process(CanIOWC,CanAction,CanBD)
 begin
  if CanIOWC = '0' then
     Case CanAction is
        when ClearRegA =>
           ClearA <= CanBD(3 downto 0);
        when Others =>
           ClearA <= "0000";
     end Case;
  else
     ClearA <= "0000";
  end if;
 end process;
--------------------------------------------------------------------
---------------------------- RegA1----------------------------
--------------------------------------------------------------------
RegA1Drv:process(Herstel,SubBlock,NCSOut,SubAdrs,SubEndAdrs,ClearA,DoSubWrite)
 begin
  if Herstel = '1' or (ClearA(0) = '1') then
    RegA1 <= '0';
  elsif NCSout ' event and NCSOut = '0' then
     if (SubAdrs = SubEndAdrs) and (DoSubWrite) then
        Case SubBlock is
           when "00" =>
              RegA1 <= '1';
```

```
            when Others =>
          end Case;
       end if;
    end if;
 end process;
-------------------------------------------------------------------------
-------------------------------- RegA2----------------------------------
-------------------------------------------------------------------------
RegA2Drv:process(Herstel,SubBlock,NCSOut,SubAdrs,SubEndAdrs,ClearA,DoSubWrite)
 begin
   if Herstel = '1' or (ClearA(1) = '1') then
      RegA2 <= '0';
   elsif NCSout ' event and NCSOut = '0' then
      if (SubAdrs = SubEndAdrs) and (DoSubWrite) then
         Case SubBlock is
            when "01" =>
               RegA2 <= '1';
            when Others =>
         end Case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
-------------------------------- RegA3----------------------------------
-------------------------------------------------------------------------
RegA3Drv:process(Herstel,SubBlock,NCSOut,SubAdrs,SubEndAdrs,ClearA,DoSubWrite)
 begin
   if Herstel = '1' or (ClearA(2) = '1') then
      RegA3 <= '0';
   elsif NCSout ' event and NCSOut = '0' then
      if (SubAdrs = SubEndAdrs) and (DoSubWrite) then
         Case SubBlock is
            when "10" =>
               RegA3 <= '1';
            when Others =>
         end Case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
-------------------------------- RegA4----------------------------------
-------------------------------------------------------------------------
RegA4Drv:process(Herstel,SubBlock,NCSOut,SubAdrs,SubEndAdrs,ClearA,DoSubWrite)
 begin
   if Herstel = '1' or (ClearA(3) = '1') then
      RegA4 <= '0';
   elsif NCSout ' event and NCSOut = '0' then
      if (SubAdrs = SubEndAdrs) and (DoSubWrite) then
         Case SubBlock is
            when "11" =>
               RegA4 <= '1';
            when Others =>
         end Case;
      end if;
   end if;
 end process;
-------------------------------------------------------------------------
-------------------------------- ClearB---------------------------------
-------------------------------------------------------------------------
ClearBDrv:process(Herstel,SubBlock,NCSOut,SubAdrs,SubEndAdrs,DoSubRead)
 begin
   if NCSout ' event and NCSOut = '0' then
      if (SubAdrs = SubEndAdrs) and (DoSubRead) then
         Case SubBlock is
            when "00" =>
               ClearB <= "0001";
            when "01" =>
               ClearB <= "0010";
            when "10" =>
               ClearB <= "0100";
            when "11" =>
               ClearB <= "1000";
            when Others =>
         end Case;
      else
```

```
        ClearB <= "0000";
      end if;
   end if;
end process;
-------------------------------------------------------------------------
---------------------------------- RegB1---------------------------------
-------------------------------------------------------------------------
RegB1Drv:process(Herstel,CanIOWC,CanAction,ClearB)
 begin
  if ((Herstel = '1') or (ClearB(0) = '1')) then
     RegB1 <= '0';
  elsif CanIOWC = '0' then
     Case CanAction is
        when SetRegB =>
           RegB1 <= (RegB1 or CanBD(0));
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------
---------------------------------- RegB2---------------------------------
-------------------------------------------------------------------------
RegB2Drv:process(Herstel,CanIOWC,CanAction,ClearB)
 begin
  if ((Herstel = '1') or (ClearB(1) = '1')) then
     RegB2 <= '0';
  elsif CanIOWC = '0' then
     Case CanAction is
        when SetRegB =>
           RegB2 <= (RegB2 or CanBD(1));
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------
---------------------------------- RegB3---------------------------------
-------------------------------------------------------------------------
RegB3Drv:process(Herstel,CanIOWC,CanAction,ClearB)
 begin
  if ((Herstel = '1') or (ClearB(2) = '1')) then
     RegB3 <= '0';
  elsif CanIOWC = '0' then
     Case CanAction is
        when SetRegB =>
           RegB3 <= (RegB3 or CanBD(2));
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------
---------------------------------- RegB4---------------------------------
-------------------------------------------------------------------------
RegB4Drv:process(Herstel,CanIOWC,CanAction,ClearB)
 begin
  if ((Herstel = '1') or (ClearB(3) = '1')) then
     RegB4 <= '0';
  elsif CanIOWC = '0' then
     Case CanAction is
        when SetRegB =>
           RegB4 <= (RegB4 or CanBD(3));
        when Others =>
     end Case;
  end if;
 end process;
-------------------------------------------------------------------------
---------------------------------- SubBlock-----------------------------
-------------------------------------------------------------------------
SubBlockDrv:process(SubAction,SubBd)
 begin
  if (SubIOWC = '0') then
     Case SubAction is
        when GetBlock =>
           SubBlock <= SubBd(1 downto 0);
        when Others =>
        end Case;
```

```
  end if;
 end process;
--------------------------------------------------------------------
---------------------------------- CanBlock-------------------------------
--------------------------------------------------------------------
CanBlockDrv:process(CanAction,CanBd)
 begin
  if (CanIOWC = '0') then
     Case CanAction is
        when GetBlock =>
           CanBlock <= CanBd(1 downto 0);
        when Others =>
     end Case;
  end if;
 end process;
--------------------------------------------------------------------
---------------------------------- SubSendCurrentSize---------------------
--------------------------------------------------------------------
SubSendCurrentSizeDrv:process(SubBlock,SizeA1,SizeA2,SizeA3,SizeA4)
 begin
  case SubBlock is
     when "00" =>
        SubSendCurrentSize <= unsigned(SizeA1);
     when "01" =>
        SubSendCurrentSize <= unsigned(SizeA2);
     when "10" =>
        SubSendCurrentSize <= unsigned(SizeA3);
     when "11" =>
        SubSendCurrentSize <= unsigned(SizeA4);
     when Others =>
  end case;
 end process;
--------------------------------------------------------------------
---------------------------------- SubReceiveCurrentSize---------------------
--------------------------------------------------------------------
SubReceiveCurrentSizeDrv:process(SubBlock,SizeB1,SizeB2,SizeB3,SizeB4)
 begin
  case SubBlock is
     when "00" =>
        SubReceiveCurrentSize <= unsigned(SizeB1);
     when "01" =>
        SubReceiveCurrentSize <= unsigned(SizeB2);
     when "10" =>
        SubReceiveCurrentSize <= unsigned(SizeB3);
     when "11" =>
        SubReceiveCurrentSize <= unsigned(SizeB4);
     when Others =>
  end case;
 end process;

--------------------------------------------------------------------
---------------------------------- SubEndAdrs-----------------------------
--------------------------------------------------------------------
SubEndAdrsDrv:process(SubRWClk)
 begin
  if (SubRWClk ' event and SubRWClk = '1') then
     case SubAction is
        when GetAdressHigh =>
           SubEndAdrs(10 downto 8) <= unsigned(SubBd(2 downto 0));
        when GetAdressLow =>
           SubEndAdrs(7 downto 0) <= unsigned(SubBd);
        when GetDestination =>
           SubEndAdrs <= SubEndAdrs + SubSendCurrentSize - 1;
        when GiveSource =>
           SubEndAdrs <= SubEndAdrs + SubReceiveCurrentSize - 1;
        when Others =>
     end case;
  end if;
 end process;
--------------------------------------------------------------------
---------------------------------- TAct-----------------------------------
--------------------------------------------------------------------
TActDrv:process(Clk,Counter2)
 begin
  if Clk ' event and Clk = '1' then
```

```
    if (Counter2 = 3) then
      Tact <= '1';
    else
      Tact <= '0';
    end if;
  end if;
 end process;
--------------------------------------------------------------------
-------------------------------- ActivateWrite-------------------------
--------------------------------------------------------------------
ActivateWriteDrv:process(Tact,Draining,Counter2)
 begin
  if (Herstel = '1') or (Tact = '1') then
    ActivateWrite <= '0';
  elsif (Draining ' event and Draining = '0') then
    ActivateWrite <= '1';
  end if;
 end process;
--------------------------------------------------------------------
-------------------------------- ActivateRead-------------------------
--------------------------------------------------------------------
ActivateReadDrv:process(Herstel,Tact,NCSOut,SubEndAdrs,DoSubRead)
 begin
  if (Herstel = '1') or (Tact = '1') then
    ActivateRead <= '0';
  elsif NCSout ' event and NCSOut = '1' then
    if (SubAdrs = SubEndAdrs) and (DoSubRead) then
      ActivateRead <= '1';
    end if;
  end if;
end process;
--------------------------------------------------------------------
--------------------------------Counter2-----------------------------
--------------------------------------------------------------------
Counter2Drv:process(NCSOut,ActivateWrite)
 begin
  if NCSOut ' event and NCSOut = '1' then
    case PsMs is
      when ReadSub1 | ReadSub2 | WriteSub1 | WriteSub2 =>
        if (ActivateWrite = '1') or (ActivateRead = '1') then
          Counter2 <= Counter2 + 1;
        else
          Counter2 <= "00";
        end if;
      when others =>
    end case;
  end if;
 end process;
--------------------------------------------------------------------
--------------------------------LastData-----------------------------
--------------------------------------------------------------------
LastDataDrv:process(NCSOut,MemData,SubAdrs,SubEndAdrs,DoSubRead)
 begin
  if NCSout ' event and NCSOut = '1' then
    if (SubAdrs = SubEndAdrs) and (DoSubRead) then
      LastData <= MemData;
    end if;
  end if;
 end process;
--------------------------------------------------------------------
end BEHAVIORAL;
configuration CFG_board_BEHAVIORAL of board is
 for BEHAVIORAL
 end for;
end CFG_board_BEHAVIORAL;
```